ACCES I/O Products USB Library

Generated by Doxygen 1.8.6

Mon Nov 28 2016 19:39:21

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 libary **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 Caveats **31**
9.1 Device resets . . . 31
9.2 Blocking-only functionality . . . 31
9.3 Configuration selection and handling . . . 31
9.4 Early transfer completion . . . 32
9.5 Zero length packets . . . 32

10 Contexts **33**

11 Application Programming Interface **35**
11.1 Functions . . . 35
11.2 Structures . . . 37
11.3 Enums . . . 38

12 Device hotplug event notification **39**
12.1 Introduction . . . 39

13 Synchronous and asynchronous device I/O **41**
13.1 Introduction . . . 41
13.2 Transfers at a logical level . . . 41
13.3 The synchronous interface . . . 41
13.4 The asynchronous interface . . . 42

14 Packets and overflows **43**
14.1 Packet abstraction . . . 43
14.2 Bulk/interrupt transfer overflows . . . 43

15 Multi-threaded applications and asynchronous I/O **45**
15.1 libusb_handle_events() from multiple threads . . . 45
15.2 The events lock . . . 46
15.3 Letting other threads do the work for you . . . 46
15.3.1 The full story . . . 47
15.3.2 Closing remarks . . . 48

16 libusb **49**

17 README **51**

18 How to run read_channels_test_java **53**

19 How to run **55**

20 How to run extcal.java **57**

21 Native Utils **59**

22 How to run read_channels_test_java **61**

23 How to run 63

24 Todo List 65

25 Deprecated List 67

26 Module Index 69

26.1 Modules . . . 69

27 Namespace Index 71

27.1 Namespace List . . . 71

28 Hierarchical Index 73

28.1 Class Hierarchy . . . 73

29 Data Structure Index 77

29.1 Data Structures . . . 77

30 File Index 83

30.1 File List . . . 83

31 Module Documentation 89

31.1 Library initialization/deinitialization . . . 89

31.1.1 Detailed Description . . . 89

31.1.2 Typedef Documentation . . . 89

libusb_context . . . 89

31.1.3 Enumeration Type Documentation . . . 90

libusb_log_level . . . 90

31.1.4 Function Documentation . . . 90

libusb_set_debug . . . 90

libusb_init . . . 90

libusb_exit . . . 91

31.2 Device handling and enumeration . . . 92

31.2.1 Detailed Description . . . 93

31.2.2 In a nutshell... . . . 93

31.2.3 Devices and device handles . . . 94

31.2.4 Device discovery and reference counting . . . 94

31.2.5 Typedef Documentation . . . 94

libusb_device . . . 94

libusb_device_handle . . . 95

31.2.6 Enumeration Type Documentation . . . 95

libusb_speed . . . 95

libusb_supported_speed . . . 95

libusb_usb_2_0_extension_attributes . . . 95

libusb_ss_usb_device_capability_attributes . . . 95

libusb_bos_type . . . 95

31.2.7 Function Documentation . . . 96

libusb_get_device_list . . . 96

libusb_free_device_list . . . 96

libusb_get_bus_number . . . 96

libusb_get_port_number . . . 96

libusb_get_port_numbers . . . 96

libusb_get_port_path . . . 97

libusb_get_parent . . . 97
libusb_get_device_address . . . 97
libusb_get_device_speed . . . 97
libusb_get_max_packet_size . . . 97
libusb_get_max_iso_packet_size . . . 98
libusb_ref_device . . . 98
libusb_unref_device . . . 98
libusb_open . . . 98
libusb_open_device_with_vid_pid . . . 99
libusb_close . . . 99
libusb_get_device . . . 99
libusb_get_configuration . . . 99
libusb_set_configuration . . . 100
libusb_claim_interface . . . 100
libusb_release_interface . . . 101
libusb_set_interface_alt_setting . . . 101
libusb_clear_halt . . . 102
libusb_reset_device . . . 103
libusb_kernel_driver_active . . . 103
libusb_detach_kernel_driver . . . 103
libusb_attach_kernel_driver . . . 104
libusb_set_auto_detach_kernel_driver . . . 104
31.3 Miscellaneous . . . 105
31.3.1 Detailed Description . . . 106
31.3.2 Macro Definition Documentation . . . 106
LIBUSB_CALL . . . 106
LIBUSB_API_VERSION . . . 106
libusb_le16_to_cpu . . . 106
31.3.3 Enumeration Type Documentation . . . 106
libusb_standard_request . . . 106
libusb_request_type . . . 107
libusb_request_recipient . . . 107
libusb_error . . . 107
libusb_capability . . . 107
31.3.4 Function Documentation . . . 108
libusb_has_capability . . . 108
libusb_error_name . . . 108
libusb_get_version . . . 108
libusb_setlocale . . . 108
libusb_strerror . . . 108
31.4 USB descriptors . . . 110
31.4.1 Detailed Description . . . 111
31.4.2 Enumeration Type Documentation . . . 111
libusb_class_code . . . 111
libusb_descriptor_type . . . 112
libusb_endpoint_direction . . . 112
libusb_transfer_type . . . 112
libusb_iso_sync_type . . . 113
libusb_iso_usage_type . . . 113

31.4.3 Function Documentation . . . 113
libusb_get_device_descriptor . . . 113
libusb_get_active_config_descriptor . . . 113
libusb_get_config_descriptor . . . 113
libusb_get_config_descriptor_by_value . . . 114
libusb_free_config_descriptor . . . 114
libusb_get_ss_endpoint_companion_descriptor . . . 114
libusb_free_ss_endpoint_companion_descriptor . . . 115
libusb_get_bos_descriptor . . . 115
libusb_free_bos_descriptor . . . 115
libusb_get_usb_2_0_extension_descriptor . . . 115
libusb_free_usb_2_0_extension_descriptor . . . 115
libusb_get_ss_usb_device_capability_descriptor . . . 116
libusb_free_ss_usb_device_capability_descriptor . . . 117
libusb_get_container_id_descriptor . . . 117
libusb_free_container_id_descriptor . . . 117
libusb_get_string_descriptor_ascii . . . 117
31.5 Device hotplug event notification . . . 118
31.5.1 Detailed Description . . . 118
31.5.2 Macro Definition Documentation . . . 118
LIBUSB_HOTPLUG_MATCH_ANY . . . 118
31.5.3 Typedef Documentation . . . 118
libusb_hotplug_callback_handle . . . 118
libusb_hotplug_callback_fn . . . 119
31.5.4 Enumeration Type Documentation . . . 119
libusb_hotplug_flag . . . 119
libusb_hotplug_event . . . 119
31.5.5 Function Documentation . . . 119
libusb_hotplug_register_callback . . . 119
libusb_hotplug_deregister_callback . . . 120
31.6 Asynchronous device I/O . . . 121
31.6.1 Detailed Description . . . 121
31.6.2 Transfer abstraction . . . 122
31.6.3 Asynchronous transfers . . . 122
Allocation . . . 122
Filling . . . 122
Submission . . . 122
Completion handling . . . 122
Deallocation . . . 122
31.6.4 Resubmission . . . 123
31.6.5 Cancellation . . . 123
31.6.6 Overflows on device-to-host bulk/interrupt endpoints . . . 123
31.6.7 Considerations for control transfers . . . 123
31.6.8 Considerations for interrupt transfers . . . 124
31.6.9 Considerations for isochronous transfers . . . 124
31.6.10 Memory caveats . . . 124
31.6.11 Fine control . . . 124
31.6.12 Event handling . . . 125
31.6.13 Using an event handling thread . . . 125

31.6.14 Enumeration Type Documentation . . . 126
libusb_transfer_status . . . 126
libusb_transfer_flags . . . 126
31.6.15 Function Documentation . . . 126
libusb_alloc_streams . . . 126
libusb_free_streams . . . 127
libusb_dev_mem_alloc . . . 127
libusb_dev_mem_free . . . 127
libusb_alloc_transfer . . . 127
libusb_free_transfer . . . 128
libusb_submit_transfer . . . 128
libusb_cancel_transfer . . . 128
libusb_transfer_set_stream_id . . . 129
libusb_transfer_get_stream_id . . . 130
void . . . 130
31.7 Polling and timing . . . 131
31.7.1 Detailed Description . . . 132
31.7.2 The simple option . . . 132
31.7.3 The more advanced option . . . 132
Notes on time-based events . . . 133
Changes in the file descriptor set . . . 133
Multi-threaded considerations . . . 133
31.7.4 Typedef Documentation . . . 133
libusb_pollfd_added_cb . . . 133
libusb_pollfd_removed_cb . . . 134
31.7.5 Function Documentation . . . 134
libusb_try_lock_events . . . 134
libusb_lock_events . . . 134
libusb_unlock_events . . . 134
libusb_event_handling_ok . . . 135
libusb_event_handler_active . . . 135
libusb_interrupt_event_handler . . . 135
libusb_lock_event_waiters . . . 135
libusb_unlock_event_waiters . . . 136
libusb_wait_for_event . . . 136
libusb_handle_events_timeout_completed . . . 136
libusb_handle_events_timeout . . . 136
libusb_handle_events . . . 137
libusb_handle_events_completed . . . 137
libusb_handle_events_locked . . . 137
libusb_pollfds_handle_timeouts . . . 138
libusb_get_next_timeout . . . 138
libusb_set_pollfd_notifiers . . . 138
libusb_get_pollfds . . . 139
libusb_free_pollfds . . . 139
31.8 Synchronous device I/O . . . 140
31.8.1 Detailed Description . . . 140
31.8.2 Function Documentation . . . 140
libusb_control_transfer . . . 140

libusb_bulk_transfer . . . 140
libusb_interrupt_transfer . . . 141

32 Namespace Documentation **143**
32.1 AIOUSB Namespace Reference . . . 143
32.1.1 Function Documentation . . . 144
operator<< . . . 144
operator<< . . . 144
operator<< . . . 144
operator<< . . . 144

33 Data Structure Documentation **145**
33.1 ad_gain_pairs Struct Reference . . . 145
33.1.1 Field Documentation . . . 145
gain . . . 145
name . . . 145
33.2 ADCConfigBlock Struct Reference . . . 145
33.2.1 Field Documentation . . . 145
device . . . 145
size . . . 146
registers . . . 146
timeout . . . 146
mux_settings . . . 146
clock_rate . . . 146
discardFirstSample . . . 146
debug . . . 146
testing . . . 146
33.3 ADRange Struct Reference . . . 146
33.3.1 Field Documentation . . . 146
minVolts . . . 146
range . . . 146
33.4 AI16_DataPoint Class Reference . . . 146
33.4.1 Detailed Description . . . 147
33.4.2 Constructor & Destructor Documentation . . . 147
AI16_DataPoint . . . 147
33.4.3 Member Function Documentation . . . 147
getChannel . . . 147
getRange . . . 147
getRangeText . . . 147
isDifferentialMode . . . 147
getCounts . . . 148
getVolts . . . 148
toString . . . 148
33.4.4 Friends And Related Function Documentation . . . 148
AI16_DataPointArray . . . 148
AI16_DataSet . . . 148
AnalogInputSubsystem . . . 148
std::vector< AI16_DataPoint > . . . 148
33.4.5 Field Documentation . . . 148
counts . . . 148

channel . . . 148
range . . . 148
differentialMode . . . 148
33.5 AI16_DataPointArray Class Reference . . . 148
33.5.1 Constructor & Destructor Documentation . . . 148
AI16_DataPointArray . . . 148
33.6 AI16_DataSet Class Reference . . . 149
33.6.1 Detailed Description . . . 149
33.6.2 Constructor & Destructor Documentation . . . 149
AI16_DataSet . . . 149
∼AI16_DataSet . . . 149
33.6.3 Member Function Documentation . . . 150
getSubsystem . . . 150
getPoints . . . 150
getTimeStamp . . . 150
getCalMode . . . 150
getTriggerMode . . . 150
getOverSample . . . 150
isDiscardFirstSample . . . 151
print . . . 151
33.6.4 Friends And Related Function Documentation . . . 151
AnalogInputSubsystem . . . 151
33.6.5 Field Documentation . . . 151
subsystem . . . 151
points . . . 151
timeStamp . . . 151
calMode . . . 151
triggerMode . . . 151
overSample . . . 151
discardFirstSample . . . 151
33.7 AI16_InputRange Class Reference . . . 151
33.7.1 Constructor & Destructor Documentation . . . 152
AI16_InputRange . . . 152
AI16_InputRange . . . 152
∼AI16_InputRange . . . 152
33.7.2 Member Function Documentation . . . 152
setRange . . . 152
33.7.3 Friends And Related Function Documentation . . . 152
AnalogInputSubsystem . . . 152
AI16_DataPoint . . . 152
33.8 aio_callback_args Struct Reference . . . 152
33.8.1 Field Documentation . . . 152
transfer . . . 152
aiocb . . . 152
33.9 aio_channel_range Struct Reference . . . 152
33.9.1 Field Documentation . . . 153
start . . . 153
end . . . 153
gain . . . 153

33.10 aio_counts_converter Struct Reference . . . 153
33.10.1 Field Documentation . . . 153
num_oversamples . . . 153
num_channels . . . 153
num_scans . . . 153
unit_size . . . 153
scan_count . . . 153
channel_count . . . 153
os_count . . . 153
converted_count . . . 153
sum . . . 153
buf . . . 153
continue_conversion . . . 153
gain_ranges . . . 154
Convert . . . 154
ConvertFifo . . . 154
discardFirstSample . . . 154
33.11 aio_either_val Struct Reference . . . 154
33.11.1 Field Documentation . . . 154
number . . . 154
object . . . 154
33.12 aio_ret_value Struct Reference . . . 154
33.12.1 Field Documentation . . . 154
left . . . 154
errmsg . . . 154
right . . . 154
type . . . 154
size . . . 154
33.13 AIOArgument Struct Reference . . . 154
33.13.1 Field Documentation . . . 155
threaded . . . 155
debug . . . 155
size . . . 155
actual_size . . . 155
config . . . 155
33.14 AIOArguments Struct Reference . . . 155
33.14.1 Field Documentation . . . 155
device_args . . . 155
number_arguments . . . 155
33.15 AIOBuf Struct Reference . . . 155
33.15.1 Field Documentation . . . 155
size . . . 155
endpos . . . 155
type . . . 155
defined . . . 156
33.16 aiobuf_iterator Struct Reference . . . 156
33.16.1 Field Documentation . . . 156
buf . . . 156
loc . . . 156

next . . . 156
33.17 AIOChannelMask Struct Reference . . . 156
33.17.1 Field Documentation . . . 156
signal_indices . . . 156
signal_index . . . 156
active_signals . . . 156
signals . . . 156
number_signals . . . 156
pos . . . 156
size . . . 156
strrep . . . 156
strrepsmall . . . 156
33.18 AIOChannelRangeTmp Struct Reference . . . 157
33.18.1 Field Documentation . . . 157
start_channel . . . 157
end_channel . . . 157
gaincode . . . 157
33.19 AIOCmd Struct Reference . . . 157
33.19.1 Field Documentation . . . 157
stop_scan . . . 157
stop_scan_arg . . . 157
channel . . . 157
num_scans . . . 157
num_channels . . . 157
num_samples . . . 157
33.20 AIOCommandLineOptions Struct Reference . . . 157
33.20.1 Field Documentation . . . 158
pass_through . . . 158
num_scans . . . 158
default_num_scans . . . 158
num_channels . . . 158
default_num_channels . . . 158
num_oversamples . . . 158
default_num_oversamples . . . 158
gain_code . . . 158
clock_rate . . . 158
default_clock_rate . . . 158
outfile . . . 158
reset . . . 158
debug_level . . . 158
number_ranges . . . 158
verbose . . . 158
start_channel . . . 159
default_start_channel . . . 159
end_channel . . . 159
default_end_channel . . . 159
index . . . 159
block_size . . . 159
with_timing . . . 159

slow_acquire . . . 159
buffer_size . . . 159
rate_limit . . . 159
physical . . . 159
counts . . . 159
calibration . . . 159
repeat_number . . . 159
aiobuf_json . . . 159
default_aiobuf_json . . . 159
adcconfig_json . . . 159
ranges . . . 159
33.21 AIOContinuousBuf Struct Reference . . . 159
33.21.1 Detailed Description . . . 160
33.21.2 Field Documentation . . . 160
callback . . . 160
worker . . . 160
lock . . . 160
tattr . . . 160
work . . . 160
DeviceIndex . . . 160
fifo . . . 160
buffer . . . 160
countsbuf . . . 160
unit_size . . . 160
hz . . . 160
base_size . . . 160
size . . . 160
num_oversamples . . . 160
num_channels . . . 160
num_scans . . . 160
scans_read . . . 160
start_scanning . . . 160
block_size . . . 160
bytes_processed . . . 161
counter_control . . . 161
timeout . . . 161
exitcode . . . 161
testing . . . 161
debug . . . 161
mask . . . 161
status . . . 161
type . . . 161
PushN . . . 161
PopN . . . 161
33.22 AIODeviceInfo Struct Reference . . . 161
33.22.1 Field Documentation . . . 161
PID . . . 161
NameSize . . . 161
Name . . . 161

DIOBytes . . . 161
Counters . . . 161
33.23 AIODeviceQuery Struct Reference . . . 161
33.23.1 Field Documentation . . . 162
productID . . . 162
nameSize . . . 162
name . . . 162
numDIOBytes . . . 162
numCounters . . . 162
index . . . 162
33.24 aioerror Struct Reference . . . 162
33.24.1 Field Documentation . . . 162
retval . . . 162
error_message . . . 162
33.25 AIOFifo Struct Reference . . . 163
33.25.1 Detailed Description . . . 163
33.25.2 Field Documentation . . . 163
AIO_FIFO_INTERFACE . . . 163
LOCKING_MECHANISM . . . 163
33.26 AIOGainRange Struct Reference . . . 163
33.26.1 Field Documentation . . . 163
min . . . 163
max . . . 163
33.27 AIOProductGroup Struct Reference . . . 163
33.27.1 Detailed Description . . . 164
33.28 AIOProductRange Struct Reference . . . 164
33.28.1 Detailed Description . . . 164
33.29 aiousb_libusb_args Struct Reference . . . 164
33.29.1 Field Documentation . . . 164
dev . . . 164
handle . . . 164
deviceDesc . . . 164
33.30 AIOUSBDevice Struct Reference . . . 164
33.30.1 Field Documentation . . . 166
usb_device . . . 166
bOpen . . . 166
deviceIndex . . . 166
isInit . . . 166
PID . . . 166
DIOConfigBits . . . 166
discardFirstSample . . . 166
commTimeout . . . 166
miscClockHz . . . 166
ProductID . . . 166
DIOBytes . . . 166
Counters . . . 166
Tristates . . . 166
bGateSelectable . . . 166
RootClock . . . 166

bGetName . . . 166
ConfigBytes . . . 166
ImmDACs . . . 166
bDACStream . . . 166
bDACDIOStream . . . 166
bDACSlowWaveStream . . . 166
bDACDIOClock . . . 166
DACsUsed . . . 166
bADCStream . . . 166
ADCChannels . . . 166
ADCMUXChannels . . . 166
RangeShift . . . 166
ADCChannelsPerGroup . . . 166
bDIOStream . . . 167
StreamingBlockSize . . . 167
bDIODebounce . . . 167
bDIOSPI . . . 167
bSetCustomClocks . . . 167
WDGBytes . . . 167
bClearFIFO . . . 167
ImmADCs . . . 167
bDACBoardRange . . . 167
bDACChannelCal . . . 167
FlashSectors . . . 167
bDACOpen . . . 167
bDACClosing . . . 167
bDACAborting . . . 167
bDACStarted . . . 167
DACData . . . 167
PendingDACData . . . 167
hDACDataMutex . . . 167
hDACDataSem . . . 167
bDIOOpen . . . 167
bDIORead . . . 167
bDeviceWasHere . . . 167
LastDIOData . . . 167
cachedName . . . 167
cachedSerialNumber . . . 167
cachedConfigBlock . . . 167
workerBusy . . . 167
workerStatus . . . 167
workerResult . . . 168
FastITConfig . . . 168
FastITBakConfig . . . 168
FastITConfig_size . . . 168
ADBuf . . . 168
ADBuf_size . . . 168
testing . . . 168
valid . . . 168

33.31 aiousboption Struct Reference . . . 168
33.32 AIOWDGConfig Struct Reference . . . 168
33.32.1 Field Documentation . . . 168
bufsize . . . 168
L . . . 168
wdgbuf . . . 168
timeout . . . 168
33.33 AnalogInputSubsystem Class Reference . . . 168
33.33.1 Detailed Description . . . 172
33.33.2 Constructor & Destructor Documentation . . . 172
AnalogInputSubsystem . . . 172
~AnalogInputSubsystem . . . 172
33.33.3 Member Function Documentation . . . 172
setScanRange . . . 172
print . . . 172
getNumChannels . . . 172
getNumMUXChannels . . . 172
getChannelsPerGroup . . . 172
isAutoCalPresent . . . 173
getRangeText . . . 173
isAutoConfig . . . 173
setAutoConfig . . . 173
readConfig . . . 174
writeConfig . . . 174
isDiscardFirstSample . . . 174
setDiscardFirstSample . . . 174
getCalMode . . . 175
setCalMode . . . 175
getTriggerMode . . . 175
setTriggerMode . . . 175
getRange . . . 176
getRange . . . 176
setRange . . . 176
setRange . . . 176
isDifferentialMode . . . 177
isDifferentialMode . . . 177
setDifferentialMode . . . 177
setDifferentialMode . . . 178
setRangeAndDiffMode . . . 178
setRangeAndDiffMode . . . 178
setRangeAndDiffMode . . . 179
getOverSample . . . 179
setOverSample . . . 179
setCalibrationTable . . . 179
setCalibrationTable . . . 179
getStreamingBlockSize . . . 180
setStreamingBlockSize . . . 180
getClock . . . 180
setClock . . . 180

calibrate . . . 180
calibrate . . . 181
read . . . 181
readCounts . . . 182
readCounts . . . 182
readVolts . . . 182
readVolts . . . 182
readBulkStart . . . 183
readBulkSamplesAvailable . . . 183
readBulkNext . . . 183
clearFIFO . . . 183
countsToVolts . . . 184
countsToVolts . . . 184
voltsToCounts . . . 184
voltsToCounts . . . 185
33.33.4 Friends And Related Function Documentation . . . 185
USB_AI16_Family . . . 185
33.33.5 Field Documentation . . . 185
CAL_MODE_NORMAL . . . 185
CAL_MODE_GROUND . . . 185
CAL_MODE_REFERENCE . . . 185
TRIG_MODE_CTR0_EXT . . . 185
TRIG_MODE_FALLING_EDGE . . . 185
TRIG_MODE_SCAN . . . 185
TRIG_MODE_EXTERNAL . . . 185
TRIG_MODE_TIMER . . . 186
RANGE_0_10V . . . 186
RANGE_10V . . . 186
RANGE_0_5V . . . 186
RANGE_5V . . . 186
RANGE_0_2V . . . 186
RANGE_2V . . . 186
RANGE_0_1V . . . 186
RANGE_1V . . . 186
MIN_COUNTS . . . 186
MAX_COUNTS . . . 186
CAL_TABLE_WORDS . . . 186
RANGE_TEXT . . . 186
NUM_CONFIG_REGISTERS . . . 187
NUM_MUX_CONFIG_REGISTERS . . . 187
NUM_GAIN_CODE_REGISTERS . . . 187
REG_GAIN_CODE . . . 187
REG_CAL_MODE . . . 187
REG_TRIG_MODE . . . 187
REG_START_END . . . 187
REG_OVERSAMPLE . . . 187
REG_MUX_START_END . . . 187
DIFFERENTIAL_MODE . . . 187
MAX_OVERSAMPLE . . . 187

TRIG_MODE_VALID_MASK . . . 187
AUTO_CAL_UNKNOWN . . . 187
AUTO_CAL_NOT_PRESENT . . . 187
AUTO_CAL_PRESENT . . . 187
MAX_CHANNELS . . . 187
numChannels . . . 187
numMUXChannels . . . 187
channelsPerGroup . . . 187
configBlockSize . . . 187
autoCalFeature . . . 187
inputRange . . . 187
differentialMode . . . 187
calMode . . . 187
triggerMode . . . 187
startChannel . . . 187
endChannel . . . 187
overSample . . . 187
readBulkBuffer . . . 187
readBulkSamplesRequested . . . 187
readBulkSamplesRetrieved . . . 187
autoConfig . . . 188
33.34 AnalogIORange Class Reference . . . 188
33.34.1 Detailed Description . . . 188
33.34.2 Constructor & Destructor Documentation . . . 188
AnalogIORange . . . 188
AnalogIORange . . . 188
~AnalogIORange . . . 189
33.34.3 Member Function Documentation . . . 189
getRange . . . 189
setRange . . . 189
setCountRange . . . 189
setVoltRange . . . 189
countsToVolts . . . 189
voltsToCounts . . . 190
33.34.4 Field Documentation . . . 190
range . . . 190
minCounts . . . 190
maxCounts . . . 190
rangeCounts . . . 190
minVolts . . . 190
maxVolts . . . 190
rangeVolts . . . 190
33.35 AnalogOutputSubsystem Class Reference . . . 190
33.35.1 Detailed Description . . . 191
33.35.2 Constructor & Destructor Documentation . . . 191
AnalogOutputSubsystem . . . 191
~AnalogOutputSubsystem . . . 191
33.35.3 Member Function Documentation . . . 191
print . . . 191

getNumChannels . . . 191
writeCounts . . . 191
writeCounts . . . 191
33.35.4 Field Documentation . . . 192
numChannels . . . 192
minCounts . . . 192
maxCounts . . . 192
33.36 AO16_AnalogOutputSubsystem Class Reference . . . 192
33.36.1 Detailed Description . . . 193
33.36.2 Constructor & Destructor Documentation . . . 193
AO16_AnalogOutputSubsystem . . . 193
~AO16_AnalogOutputSubsystem . . . 193
33.36.3 Member Function Documentation . . . 193
getRangeText . . . 193
getRange . . . 193
setRange . . . 194
writeVolts . . . 194
writeVolts . . . 194
countsToVolts . . . 194
voltsToCounts . . . 194
33.36.4 Friends And Related Function Documentation . . . 195
USB_AO16_Family . . . 195
33.36.5 Field Documentation . . . 195
RANGE_0_5V . . . 195
RANGE_5V . . . 195
RANGE_0_10V . . . 195
RANGE_10V . . . 195
MIN_COUNTS . . . 195
MAX_COUNTS . . . 195
RANGE_TEXT . . . 195
outputRange . . . 195
33.37 AO16_OutputRange Class Reference . . . 195
33.37.1 Constructor & Destructor Documentation . . . 196
AO16_OutputRange . . . 196
AO16_OutputRange . . . 196
~AO16_OutputRange . . . 196
33.37.2 Member Function Documentation . . . 196
setRange . . . 196
33.37.3 Friends And Related Function Documentation . . . 196
AO16_AnalogOutputSubsystem . . . 196
33.38 BLooper Class Reference . . . 196
33.39 BoolArray Class Reference . . . 196
33.39.1 Constructor & Destructor Documentation . . . 196
BoolArray . . . 196
33.40 BulkAcquireWorkerParams Struct Reference . . . 197
33.40.1 Field Documentation . . . 197
DeviceIndex . . . 197
BufSize . . . 197
pBuf . . . 197

33.41 channel_range Struct Reference . . . 197
33.41.1 Field Documentation . . . 197
start_channel . . . 197
end_channel . . . 197
gaincode . . . 197
33.42 cJSON Struct Reference . . . 197
33.42.1 Field Documentation . . . 198
next . . . 198
prev . . . 198
child . . . 198
type . . . 198
valuestring . . . 198
valueint . . . 198
valuedouble . . . 198
string . . . 198
33.43 cJSON_Hooks Struct Reference . . . 198
33.43.1 Field Documentation . . . 198
malloc_fn . . . 198
free_fn . . . 198
33.44 command_block_wrapper Struct Reference . . . 198
33.44.1 Field Documentation . . . 198
dCBWSignature . . . 198
dCBWTag . . . 198
dCBWDataTransferLength . . . 198
bmCBWFlags . . . 198
bCBWLUN . . . 198
bCBWCBLength . . . 199
CBWCB . . . 199
33.45 command_status_wrapper Struct Reference . . . 199
33.45.1 Field Documentation . . . 199
dCSWSignature . . . 199
dCSWTag . . . 199
dCSWDataResidue . . . 199
bCSWStatus . . . 199
33.46 config_options Struct Reference . . . 199
33.46.1 Field Documentation . . . 199
targetSerialNumber . . . 199
framePoints . . . 199
buffer_size . . . 199
clock_rate . . . 199
number_channels . . . 199
write_clock_rate . . . 199
33.47 configuration Struct Reference . . . 199
33.47.1 Field Documentation . . . 200
type . . . 200
timeout . . . 200
discard_first_sample . . . 200
device_index . . . 200
number_scans . . . 200

calibration . . . 200
scan_type . . . 200
calibration_file . . . 200
debug . . . 200
output_file . . . 200
file_handle . . . 200
file_name . . . 200
configure . . . 200
run . . . 200
33.48 Counter Class Reference . . . 200
33.48.1 Detailed Description . . . 201
33.48.2 Constructor & Destructor Documentation . . . 201
Counter . . . 201
33.48.3 Member Function Documentation . . . 201
getDeviceIndex . . . 201
setMode . . . 201
setCount . . . 202
setModeAndCount . . . 202
readCount . . . 202
readCountAndStatus . . . 202
readCountAndSetModeAndCount . . . 203
33.48.4 Friends And Related Function Documentation . . . 203
CounterSubsystem . . . 203
33.48.5 Field Documentation . . . 203
MODE_TERMINAL_COUNT . . . 203
MODE_ONE_SHOT . . . 203
MODE_RATE_GENERATOR . . . 203
MODE_SQUARE_WAVE . . . 203
MODE_SW_TRIGGERED . . . 203
MODE_HW_TRIGGERED . . . 203
parent . . . 203
counterIndex . . . 203
33.49 CounterList Class Reference . . . 204
33.50 CounterSubsystem Class Reference . . . 204
33.50.1 Detailed Description . . . 204
33.50.2 Constructor & Destructor Documentation . . . 205
CounterSubsystem . . . 205
~CounterSubsystem . . . 205
33.50.3 Member Function Documentation . . . 205
print . . . 205
getNumCounterBlocks . . . 205
getNumCounters . . . 205
getCounter . . . 205
readCounts . . . 205
selectGate . . . 206
startClock . . . 206
stopClock . . . 206
33.50.4 Friends And Related Function Documentation . . . 207
Counter . . . 207

USB_AI16_Family . . . 207
USB_CTR_15_Family . . . 207
USB_DIO_32_Family . . . 207
33.50.5 Field Documentation . . . 207
COUNTERS_PER_BLOCK . . . 207
numCounterBlocks . . . 207
numCounters . . . 207
counters . . . 207
33.51 CStringArray Struct Reference . . . 207
33.52 DA12_AnalogOutputSubsystem Class Reference . . . 207
33.52.1 Detailed Description . . . 208
33.52.2 Constructor & Destructor Documentation . . . 208
DA12_AnalogOutputSubsystem . . . 208
~DA12_AnalogOutputSubsystem . . . 208
33.52.3 Member Function Documentation . . . 208
getRangeText . . . 208
getRange . . . 209
getRange . . . 209
setRange . . . 209
setRange . . . 210
setRange . . . 210
writeVolts . . . 210
writeVolts . . . 210
countsToVolts . . . 211
voltsToCounts . . . 211
33.52.4 Friends And Related Function Documentation . . . 211
USB_DA12_8A_Family . . . 211
USB_DA12_8E_Family . . . 211
33.52.5 Field Documentation . . . 211
RANGE_0_2_5V . . . 211
RANGE_2_5V . . . 211
RANGE_0_5V . . . 211
RANGE_5V . . . 212
RANGE_0_10V . . . 212
RANGE_10V . . . 212
MIN_COUNTS . . . 212
MAX_COUNTS . . . 212
RANGE_TEXT . . . 212
outputRange . . . 212
33.53 DA12_OutputRange Class Reference . . . 212
33.53.1 Constructor & Destructor Documentation . . . 213
DA12_OutputRange . . . 213
DA12_OutputRange . . . 213
~DA12_OutputRange . . . 213
33.53.2 Member Function Documentation . . . 213
setRange . . . 213
33.53.3 Friends And Related Function Documentation . . . 213
DA12_AnalogOutputSubsystem . . . 213
33.54 darwin_cached_device Struct Reference . . . 213

33.54.1 Field Documentation . . . 213
list . . . 213
dev_descriptor . . . 213
location . . . 213
parent_session . . . 213
session . . . 214
address . . . 214
sys_path . . . 214
device . . . 214
open_count . . . 214
first_config . . . 214
active_config . . . 214
port . . . 214
can_enumerate . . . 214
refcount . . . 214
33.55darwin_device_handle_priv Struct Reference . . . 214
33.55.1 Field Documentation . . . 214
is_open . . . 214
cfSource . . . 214
interfaces . . . 214
33.56darwin_device_priv Struct Reference . . . 214
33.56.1 Field Documentation . . . 215
dev . . . 215
33.57darwin_device_handle_priv::darwin_interface Struct Reference . . . 215
33.57.1 Field Documentation . . . 215
interface . . . 215
num_endpoints . . . 215
cfSource . . . 215
frames . . . 215
endpoint_addrs . . . 215
33.58darwin_transfer_priv Struct Reference . . . 215
33.58.1 Field Documentation . . . 215
isoc_framelist . . . 215
num_iso_packets . . . 215
req . . . 215
result . . . 215
size . . . 215
33.59device_priv Struct Reference . . . 216
33.59.1 Field Documentation . . . 216
devnode . . . 216
fd . . . 216
cdesc . . . 216
ddesc . . . 216
devname . . . 216
33.60DeviceInfo Struct Reference . . . 216
33.60.1 Field Documentation . . . 216
outputMask . . . 216
readBuffer . . . 216
writeBuffer . . . 216

name . . . 216
productID . . . 216
nameSize . . . 216
numDIOBytes . . . 216
numCounters . . . 216
serialNumber . . . 216
index . . . 216
33.61 DeviceProperties Struct Reference . . . 217
33.61.1 Detailed Description . . . 217
33.61.2 Field Documentation . . . 217
Name . . . 217
SerialNumber . . . 217
ProductID . . . 217
DIOPorts . . . 217
Counters . . . 217
Tristates . . . 217
RootClock . . . 218
DACChannels . . . 218
ADCChannels . . . 218
ADCMUXChannels . . . 218
ADCChannelsPerGroup . . . 218
33.62 DeviceSubsystem Class Reference . . . 218
33.62.1 Detailed Description . . . 218
33.62.2 Constructor & Destructor Documentation . . . 218
DeviceSubsystem . . . 218
~DeviceSubsystem . . . 218
33.62.3 Member Function Documentation . . . 218
getDeviceIndex . . . 218
print . . . 219
getParent . . . 219
33.62.4 Field Documentation . . . 219
parent . . . 219
33.63 devlink_cbarg Struct Reference . . . 219
33.63.1 Field Documentation . . . 219
nargs . . . 219
myself . . . 219
minor . . . 219
33.64 DigitalIOSubsystem Class Reference . . . 219
33.64.1 Detailed Description . . . 220
33.64.2 Constructor & Destructor Documentation . . . 220
DigitalIOSubsystem . . . 220
~DigitalIOSubsystem . . . 220
33.64.3 Member Function Documentation . . . 220
bitsToBytes . . . 220
bytesToBits . . . 220
print . . . 220
getNumPorts . . . 221
getNumChannels . . . 221
getNumTristateGroups . . . 221

getNumTristateChannels . . . 221
configure . . . 221
configure . . . 222
getConfiguration . . . 222
read . . . 222
read . . . 223
write . . . 223
write . . . 223
33.64.4 Friends And Related Function Documentation . . . 223
USB_AI16_Family . . . 223
USB_AO16_Family . . . 223
USB_DIO_Family . . . 224
USB_DIO_16_Family . . . 224
USB_DIO_32_Family . . . 224
33.64.5 Field Documentation . . . 224
numPorts . . . 224
numChannels . . . 224
numTristateGroups . . . 224
numTristateChannels . . . 224
writeValues . . . 224
33.65 DIOBuf Struct Reference . . . 224
33.65.1 Detailed Description . . . 224
33.65.2 Field Documentation . . . 225
size . . . 225
buffer . . . 225
strbuf . . . 225
strbuf_size . . . 225
33.66 DIOStreamSubsystem Class Reference . . . 225
33.66.1 Detailed Description . . . 226
33.66.2 Constructor & Destructor Documentation . . . 226
DIOStreamSubsystem . . . 226
~DIOStreamSubsystem . . . 226
33.66.3 Member Function Documentation . . . 226
print . . . 226
getStreamingBlockSize . . . 226
setStreamingBlockSize . . . 226
getClock . . . 226
setClock . . . 227
stopClock . . . 227
open . . . 227
close . . . 227
read . . . 228
write . . . 228
clearFIFO . . . 228
33.66.4 Friends And Related Function Documentation . . . 228
USB_DIO_16_Family . . . 228
33.66.5 Field Documentation . . . 228
clockHz . . . 228
33.67 discovered_devs Struct Reference . . . 229

33.67.1 Field Documentation . . . 229
len . . . 229
capacity . . . 229
devices . . . 229
33.68 DoubleArray Class Reference . . . 229
33.68.1 Constructor & Destructor Documentation . . . 229
DoubleArray . . . 229
33.69 driver_lookup Struct Reference . . . 229
33.69.1 Field Documentation . . . 229
list . . . 229
reg_prop . . . 229
designation . . . 229
33.70 endpoint Struct Reference . . . 230
33.70.1 Field Documentation . . . 230
datafd . . . 230
statfd . . . 230
33.71 Error Class Reference . . . 230
33.71.1 Constructor & Destructor Documentation . . . 230
Error . . . 230
Error . . . 230
33.71.2 Member Function Documentation . . . 230
what . . . 230
33.71.3 Field Documentation . . . 230
message . . . 230
33.72 fx_known_device Struct Reference . . . 230
33.72.1 Field Documentation . . . 231
vid . . . 231
pid . . . 231
type . . . 231
designation . . . 231
33.73 handle_priv Struct Reference . . . 231
33.73.1 Field Documentation . . . 231
endpoints . . . 231
33.74 hid_device_priv Struct Reference . . . 231
33.74.1 Field Documentation . . . 231
vid . . . 231
pid . . . 231
config . . . 231
nb_interfaces . . . 231
uses_report_ids . . . 231
input_report_size . . . 231
output_report_size . . . 231
feature_report_size . . . 231
string . . . 232
string_index . . . 232
33.75 HIDD_ATTRIBUTES Struct Reference . . . 232
33.75.1 Field Documentation . . . 232
Size . . . 232
VendorID . . . 232

ProductID . . . 232
VersionNumber . . . 232
33.76 HIDP_CAPS Struct Reference . . . 232
33.76.1 Field Documentation . . . 233
Usage . . . 233
UsagePage . . . 233
InputReportByteLength . . . 233
OutputReportByteLength . . . 233
FeatureReportByteLength . . . 233
Reserved . . . 233
NumberLinkCollectionNodes . . . 233
NumberInputButtonCaps . . . 233
NumberInputValueCaps . . . 233
NumberInputDataIndices . . . 233
NumberOutputButtonCaps . . . 233
NumberOutputValueCaps . . . 233
NumberOutputDataIndices . . . 233
NumberFeatureButtonCaps . . . 233
NumberFeatureValueCaps . . . 233
NumberFeatureDataIndices . . . 233
33.77 htab_entry Struct Reference . . . 233
33.77.1 Field Documentation . . . 233
used . . . 233
str . . . 233
33.78 IllegalArgumentException Class Reference . . . 233
33.78.1 Detailed Description . . . 234
33.78.2 Constructor & Destructor Documentation . . . 234
IllegalArgumentException . . . 234
33.79 imaxdiv_t Struct Reference . . . 234
33.79.1 Detailed Description . . . 234
33.79.2 Field Documentation . . . 234
quot . . . 234
rem . . . 234
33.80 IntArray Class Reference . . . 234
33.80.1 Constructor & Destructor Documentation . . . 234
IntArray . . . 234
33.81 interface_handle_t Struct Reference . . . 234
33.81.1 Field Documentation . . . 235
dev_handle . . . 235
api_handle . . . 235
33.82 libusb_bos_descriptor Struct Reference . . . 235
33.82.1 Detailed Description . . . 235
33.82.2 Field Documentation . . . 235
bLength . . . 235
bDescriptorType . . . 235
wTotalLength . . . 235
bNumDeviceCaps . . . 235
dev_capability . . . 236
33.83 libusb_bos_dev_capability_descriptor Struct Reference . . . 236

33.83.1 Detailed Description . . . 236
33.83.2 Field Documentation . . . 236
bLength . . . 236
bDescriptorType . . . 236
bDevCapabilityType . . . 236
dev_capability_data . . . 236
33.84 libusb_config_descriptor Struct Reference . . . 236
33.84.1 Detailed Description . . . 237
33.84.2 Field Documentation . . . 237
bLength . . . 237
bDescriptorType . . . 237
wTotalLength . . . 237
bNumInterfaces . . . 237
bConfigurationValue . . . 237
iConfiguration . . . 237
bmAttributes . . . 237
MaxPower . . . 238
interface . . . 238
extra . . . 238
extra_length . . . 238
33.85 libusb_container_id_descriptor Struct Reference . . . 238
33.85.1 Detailed Description . . . 238
33.85.2 Field Documentation . . . 238
bLength . . . 238
bDescriptorType . . . 238
bDevCapabilityType . . . 239
bReserved . . . 239
ContainerID . . . 239
33.86 libusb_context Struct Reference . . . 239
33.86.1 Field Documentation . . . 239
debug . . . 239
debug_fixed . . . 239
event_pipe . . . 239
usb_devs . . . 239
usb_devs_lock . . . 239
open_devs . . . 240
open_devs_lock . . . 240
hotplug_cbs . . . 240
hotplug_cbs_lock . . . 240
flying_transfers . . . 240
flying_transfers_lock . . . 240
fd_added_cb . . . 240
fd_removed_cb . . . 240
fd_cb_user_data . . . 240
events_lock . . . 240
event_handler_active . . . 240
event_handling_key . . . 240
event_waiters_lock . . . 240
event_waiters_cond . . . 240

event_data_lock . . . 240
event_flags . . . 240
device_close . . . 240
ipollfds . . . 240
pollfds . . . 240
pollfds_cnt . . . 240
hotplug_msgs . . . 240
completed_transfers . . . 240
list . . . 240
33.87 libusb_control_setup Struct Reference . . . 240
33.87.1 Detailed Description . . . 241
33.87.2 Field Documentation . . . 241
bmRequestType . . . 241
bRequest . . . 241
wValue . . . 241
wIndex . . . 241
wLength . . . 241
33.88 libusb_device Struct Reference . . . 241
33.88.1 Field Documentation . . . 242
lock . . . 242
refcnt . . . 242
ctx . . . 242
bus_number . . . 242
port_number . . . 242
parent_dev . . . 242
device_address . . . 242
num_configurations . . . 242
speed . . . 242
list . . . 242
session_data . . . 242
device_descriptor . . . 242
attached . . . 242
os_priv . . . 242
33.89 libusb_device_descriptor Struct Reference . . . 242
33.89.1 Detailed Description . . . 243
33.89.2 Field Documentation . . . 243
bLength . . . 243
bDescriptorType . . . 243
bcdUSB . . . 243
bDeviceClass . . . 243
bDeviceSubClass . . . 243
bDeviceProtocol . . . 243
bMaxPacketSize0 . . . 243
idVendor . . . 243
idProduct . . . 243
bcdDevice . . . 243
iManufacturer . . . 243
iProduct . . . 244
iSerialNumber . . . 244

bNumConfigurations . . . 244
33.90 libusb_device_handle Struct Reference . . . 244
33.90.1 Field Documentation . . . 244
lock . . . 244
claimed_interfaces . . . 244
list . . . 244
dev . . . 244
auto_detach_kernel_driver . . . 244
os_priv . . . 244
33.91 libusb_endpoint_descriptor Struct Reference . . . 244
33.91.1 Detailed Description . . . 245
33.91.2 Field Documentation . . . 245
bLength . . . 245
bDescriptorType . . . 245
bEndpointAddress . . . 245
bmAttributes . . . 245
wMaxPacketSize . . . 245
bInterval . . . 245
bRefresh . . . 245
bSynchAddress . . . 245
extra . . . 246
extra_length . . . 246
33.92 libusb_hid_descriptor Struct Reference . . . 246
33.92.1 Field Documentation . . . 246
bLength . . . 246
bDescriptorType . . . 246
bcdHID . . . 246
bCountryCode . . . 246
bNumDescriptors . . . 246
bClassDescriptorType . . . 246
wClassDescriptorLength . . . 246
33.93 libusb_hotplug_callback Struct Reference . . . 246
33.93.1 Detailed Description . . . 247
33.93.2 Field Documentation . . . 247
ctx . . . 247
vendor_id . . . 247
product_id . . . 247
dev_class . . . 247
flags . . . 247
events . . . 247
cb . . . 247
handle . . . 247
user_data . . . 247
needs_free . . . 247
list . . . 248
33.94 libusb_hotplug_message Struct Reference . . . 248
33.94.1 Field Documentation . . . 248
event . . . 248
device . . . 248

list . . . 248
33.95 libusb_interface Struct Reference . . . 248
33.95.1 Detailed Description . . . 248
33.95.2 Field Documentation . . . 248
altsetting . . . 248
num_altsetting . . . 249
33.96 libusb_interface_descriptor Struct Reference . . . 249
33.96.1 Detailed Description . . . 249
33.96.2 Field Documentation . . . 249
bLength . . . 249
bDescriptorType . . . 249
bInterfaceNumber . . . 249
bAlternateSetting . . . 250
bNumEndpoints . . . 250
bInterfaceClass . . . 250
bInterfaceSubClass . . . 250
bInterfaceProtocol . . . 250
iInterface . . . 250
endpoint . . . 250
extra . . . 250
extra_length . . . 250
33.97 libusb_iso_packet_descriptor Struct Reference . . . 250
33.97.1 Detailed Description . . . 250
33.97.2 Field Documentation . . . 251
length . . . 251
actual_length . . . 251
status . . . 251
33.98 libusb_pollfd Struct Reference . . . 251
33.98.1 Detailed Description . . . 251
33.98.2 Field Documentation . . . 251
fd . . . 251
events . . . 251
33.99 libusb_ss_endpoint_companion_descriptor Struct Reference . . . 251
33.99.1 Detailed Description . . . 252
33.99.2 Field Documentation . . . 252
bLength . . . 252
bDescriptorType . . . 252
bMaxBurst . . . 252
bmAttributes . . . 252
wBytesPerInterval . . . 252
33.100 libusb_ss_usb_device_capability_descriptor Struct Reference . . . 252
33.100.1 Detailed Description . . . 253
33.100.2 Field Documentation . . . 253
bLength . . . 253
bDescriptorType . . . 253
bDevCapabilityType . . . 253
bmAttributes . . . 253
wSpeedSupported . . . 253
bFunctionalitySupport . . . 253

bU1DevExitLat . . . 253
bU2DevExitLat . . . 253
33.101 libusb_testlib_ctx Struct Reference . . . 254
33.101.1 Detailed Description . . . 254
33.101.2 Field Documentation . . . 254
test_names . . . 254
test_count . . . 254
list_tests . . . 254
verbose . . . 254
old_stdout . . . 254
old_stderr . . . 254
output_file . . . 254
null_fd . . . 254
33.102 libusb_testlib_test Struct Reference . . . 254
33.102.1 Detailed Description . . . 254
33.102.2 Field Documentation . . . 254
name . . . 254
function . . . 255
33.103 libusb_transfer Struct Reference . . . 255
33.103.1 Detailed Description . . . 255
33.103.2 Field Documentation . . . 255
dev_handle . . . 255
flags . . . 255
endpoint . . . 255
type . . . 256
timeout . . . 256
status . . . 256
length . . . 256
actual_length . . . 256
callback . . . 256
user_data . . . 256
buffer . . . 256
num_iso_packets . . . 256
iso_packet_desc . . . 256
33.104 libusb_usb_2_0_extension_descriptor Struct Reference . . . 256
33.104.1 Detailed Description . . . 257
33.104.2 Field Documentation . . . 257
bLength . . . 257
bDescriptorType . . . 257
bDevCapabilityType . . . 257
bmAttributes . . . 257
33.105 libusb_version Struct Reference . . . 257
33.105.1 Detailed Description . . . 258
33.105.2 Field Documentation . . . 258
major . . . 258
minor . . . 258
micro . . . 258
nano . . . 258
rc . . . 258

describe . . . 258
33.106 linux_device_handle_priv Struct Reference . . . 258
33.106.1 Field Documentation . . . 258
fd . . . 258
fd_removed . . . 258
caps . . . 258
33.107 linux_device_priv Struct Reference . . . 259
33.107.1 Field Documentation . . . 259
sysfs_dir . . . 259
descriptors . . . 259
descriptors_len . . . 259
active_config . . . 259
33.108 linux_transfer_priv Struct Reference . . . 259
33.108.1 Field Documentation . . . 259
urbs . . . 259
iso_urbs . . . 259
"@32 . . . 259
reap_action . . . 259
num_urbs . . . 259
num_retired . . . 259
reap_status . . . 259
iso_packet_offset . . . 259
33.109 list_head Struct Reference . . . 259
33.109.1 Field Documentation . . . 260
prev . . . 260
next . . . 260
33.110 lookup Struct Reference . . . 260
33.110.1 Field Documentation . . . 260
value . . . 260
str . . . 260
strvalue . . . 260
33.111 linux_settings Struct Reference . . . 260
33.111.1 Field Documentation . . . 260
ADCChannelsPerGroup . . . 260
ADCMUXChannels . . . 260
defined . . . 260
33.112 new_aio_fifo Struct Reference . . . 260
33.112.1 Field Documentation . . . 261
AIO_FIFO_INTERFACE . . . 261
LOCKING_MECHANISM . . . 261
Push . . . 261
PushN . . . 261
Pop . . . 261
PopN . . . 261
33.113 node_args Struct Reference . . . 261
33.113.1 Field Documentation . . . 261
ctx . . . 261
discdevs . . . 261
last_ugenpath . . . 261

dlink_hdl . . . 261
33.114 OperationFailedException Class Reference . . . 261
33.114.1 Detailed Description . . . 262
33.114.2 Constructor & Destructor Documentation . . . 262
OperationFailedException . . . 262
OperationFailedException . . . 262
33.115 Option Struct Reference . . . 262
33.115.1 Field Documentation . . . 262
name . . . 262
has_arg . . . 262
flag . . . 262
val . . . 262
33.116 Options Struct Reference . . . 262
33.116.1 Field Documentation . . . 263
maxcount . . . 263
use_maxcount . . . 263
33.117 Opts Struct Reference . . . 263
33.117.1 Field Documentation . . . 263
num_scans . . . 263
default_num_scans . . . 263
num_channels . . . 263
default_num_channels . . . 263
num_oversamples . . . 264
default_num_oversamples . . . 264
gain_code . . . 264
clock_rate . . . 264
default_clock_rate . . . 264
outfile . . . 264
reset . . . 264
debug_level . . . 264
number_ranges . . . 264
verbose . . . 264
start_channel . . . 264
default_start_channel . . . 264
end_channel . . . 264
default_end_channel . . . 264
index . . . 264
block_size . . . 264
with_timing . . . 264
slow_acquire . . . 264
buffer_size . . . 264
rate_limit . . . 264
physical . . . 264
counts . . . 264
calibration . . . 264
repeat . . . 264
aiobuf_json . . . 264
default_aiobuf_json . . . 264
adcconfig_json . . . 264

ranges . . . 264
num_scans . . . 264
clock_speed . . . 264
cal_channel . . . 264
max_channels . . . 264
clock_scale . . . 264
calibration_enabled . . . 264
33.118 OutputVoltagePoint Class Reference . . . 265
33.118.1 Detailed Description . . . 265
33.118.2 Constructor & Destructor Documentation . . . 265
OutputVoltagePoint . . . 265
OutputVoltagePoint . . . 265
33.118.3 Field Documentation . . . 265
channel . . . 265
volts . . . 265
33.119 OutputVoltagePointArray Class Reference . . . 265
33.119.1 Constructor & Destructor Documentation . . . 266
OutputVoltagePointArray . . . 266
33.120 pollfd Struct Reference . . . 266
33.120.1 Field Documentation . . . 266
fd . . . 266
events . . . 266
revents . . . 266
33.121 ProductIDName Struct Reference . . . 266
33.121.1 Field Documentation . . . 266
id . . . 266
name . . . 266
33.122 ram_poke_context Struct Reference . . . 266
33.122.1 Field Documentation . . . 267
device . . . 267
mode . . . 267
total . . . 267
count . . . 267
33.123 rangelookup Struct Reference . . . 267
33.123.1 Field Documentation . . . 267
minvalue . . . 267
maxvalue . . . 267
33.124 RosterLooper Class Reference . . . 267
33.124.1 Constructor & Destructor Documentation . . . 267
RosterLooper . . . 267
33.124.2 Member Function Documentation . . . 267
Stop . . . 267
MessageReceived . . . 267
InitCheck . . . 267
33.124.3 Field Documentation . . . 267
fRoster . . . 267
fRoot . . . 268
fMessenger . . . 268
fInitCheck . . . 268

33.125 StringArray Class Reference . . . 268
33.125.1 Constructor & Destructor Documentation . . . 268
StringArray . . . 268
33.126 sunos_device_handle_priv Struct Reference . . . 268
33.126.1 Field Documentation . . . 268
altsetting . . . 268
config_index . . . 268
eps . . . 268
dpriv . . . 268
33.127 sunos_device_priv Struct Reference . . . 268
33.127.1 Field Documentation . . . 269
cfgvalue . . . 269
raw_cfgdescr . . . 269
dev_descr . . . 269
ugenpath . . . 269
phypath . . . 269
33.128 sunos_transfer_priv Struct Reference . . . 269
33.128.1 Field Documentation . . . 269
aiocb . . . 269
transfer . . . 269
33.129 tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST Struct Reference . . . 269
33.129.1 Field Documentation . . . 269
ID . . . 269
Index . . . 269
33.130 tag_USB_DK_DEVICE_ID Struct Reference . . . 269
33.130.1 Field Documentation . . . 270
DeviceID . . . 270
InstanceID . . . 270
33.131 tag_USB_DK_DEVICE_INFO Struct Reference . . . 270
33.131.1 Field Documentation . . . 270
ID . . . 270
FilterID . . . 270
Port . . . 270
Speed . . . 270
DeviceDescriptor . . . 270
33.132 tag_USB_DK_GEN_TRANSFER_RESULT Struct Reference . . . 270
33.132.1 Field Documentation . . . 270
BytesTransferred . . . 270
UsbdStatus . . . 270
33.133 tag_USB_DK_ISO_TRANSFER_RESULT Struct Reference . . . 270
33.133.1 Field Documentation . . . 271
ActualLength . . . 271
TransferResult . . . 271
33.134 tag_USB_DK_TRANSFER_REQUEST Struct Reference . . . 271
33.134.1 Field Documentation . . . 271
EndpointAddress . . . 271
Buffer . . . 271
BufferLength . . . 271
TransferType . . . 271

IsochronousPacketsArraySize . . . 271
IsochronousPacketsArray . . . 271
Result . . . 271
33.135 tag_USB_DK_TRANSFER_RESULT Struct Reference . . . 271
33.135.1 Field Documentation . . . 272
GenResult . . . 272
IsochronousResultsArray . . . 272
33.136 TestCaseSetup Class Reference . . . 272
33.136.1 Constructor & Destructor Documentation . . . 273
~TestCaseSetup . . . 273
TestCaseSetup . . . 273
TestCaseSetup . . . 273
33.136.2 Member Function Documentation . . . 273
findDevice . . . 273
findDevice . . . 273
doSomething . . . 273
setCurrentDeviceIndex . . . 273
doBulkConfigBlock . . . 273
doPreSetup . . . 273
doSetAutoCalibration . . . 273
doVerifyGroundCalibration . . . 274
doVerifyReferenceCalibration . . . 274
doDemonstrateReadVoltages . . . 274
doScanSingleChannel . . . 274
doPreReadImmediateVoltages . . . 274
doCSVReadVoltages . . . 274
doCSVWithGetChannelV . . . 274
doCleanupAfterBulk . . . 274
doDACDirect . . . 274
doDACDirectSetup . . . 274
writeBuffer . . . 274
setMaxCount . . . 274
ThrowError . . . 274
doFastITScanSetup . . . 274
doFastITScan . . . 274
doGetBuffer . . . 274
THROW_IF_ERROR . . . 274
envGetInteger . . . 274
envGetDouble . . . 274
doTestSetAutoCalibration . . . 274
doGenericVendorWrite . . . 274
doBulkAcquire . . . 274
doBulkAcquire . . . 275
doDisplayBulkResults . . . 275
resetCPU . . . 275
getVolts . . . 275
getCounts . . . 275
getGainCodes . . . 275
setupVoltageParameters . . . 275

TEST_ADC_BulkPoll . . . 275
33.136.3 Field Documentation . . . 275
productID . . . 275
nameSize . . . 275
numDIOBytes . . . 275
numCounters . . . 275
DeviceIndex . . . 275
deviceFound . . . 275
CAL_CHANNEL . . . 275
MAX_CHANNELS . . . 275
NUM_CHANNELS . . . 275
counts . . . 275
volts . . . 275
gainCodes . . . 275
configBlock . . . 275
number_oversamples . . . 275
block_size . . . 275
clock_speed . . . 275
maxcounts . . . 275
calibration_enabled . . . 275
dataBuf . . . 275
33.137 Timer_request Struct Reference . . . 276
33.137.1 Field Documentation . . . 276
tp . . . 276
event . . . 276
33.138 timespec Struct Reference . . . 276
33.138.1 Field Documentation . . . 276
tv_sec . . . 276
tv_nsec . . . 276
33.139 UCharArray Class Reference . . . 276
33.139.1 Constructor & Destructor Documentation . . . 276
UCharArray . . . 276
33.140 UKW_CONTROL_HEADER Struct Reference . . . 276
33.140.1 Field Documentation . . . 277
bmRequestType . . . 277
bRequest . . . 277
wValue . . . 277
wIndex . . . 277
wLength . . . 277
33.141 UKW_DEVICE_DESCRIPTOR Struct Reference . . . 277
33.141.1 Field Documentation . . . 277
bLength . . . 277
bDescriptorType . . . 277
bcdUSB . . . 277
bDeviceClass . . . 277
bDeviceSubClass . . . 277
bDeviceProtocol . . . 277
bMaxPacketSize0 . . . 277
idVendor . . . 277

idProduct . . . 277
bcdDevice . . . 278
iManufacturer . . . 278
iProduct . . . 278
iSerialNumber . . . 278
bNumConfigurations . . . 278
33.142 USB_AI16_Family Class Reference . . . 278
33.142.1 Detailed Description . . . 279
33.142.2 Constructor & Destructor Documentation . . . 279
USB_AI16_Family . . . 279
~USB_AI16_Family . . . 279
33.142.3 Member Function Documentation . . . 279
initialize . . . 279
getSupportedProductNames . . . 279
getSupportedProductIDs . . . 279
isSupportedProductID . . . 279
print . . . 279
adc . . . 280
dio . . . 280
ctr . . . 280
33.142.4 Friends And Related Function Documentation . . . 280
USBDeviceManager . . . 280
33.142.5 Field Documentation . . . 280
supportedProductIDs . . . 280
analogInputSubsystem . . . 280
digitalIOSubsystem . . . 280
counterSubsystem . . . 280
33.143 USB_AIO16_Family Class Reference . . . 280
33.143.1 Detailed Description . . . 281
33.143.2 Constructor & Destructor Documentation . . . 282
USB_AIO16_Family . . . 282
~USB_AIO16_Family . . . 282
33.143.3 Member Function Documentation . . . 282
initialize . . . 282
getSupportedProductNames . . . 282
getSupportedProductIDs . . . 282
isSupportedProductID . . . 282
print . . . 282
adc . . . 282
dac . . . 283
dio . . . 283
ctr . . . 283
33.143.4 Friends And Related Function Documentation . . . 283
USBDeviceManager . . . 283
33.143.5 Field Documentation . . . 283
supportedProductIDs . . . 283
analogInputSubsystem . . . 283
analogOutputSubsytem . . . 283
digitalIOSubsystem . . . 283

counterSubsystem . . . 283
33.144 USB_AO16_Family Class Reference . . . 283
33.144.1 Detailed Description . . . 284
33.144.2 Constructor & Destructor Documentation . . . 284
USB_AO16_Family . . . 284
~USB_AO16_Family . . . 284
33.144.3 Member Function Documentation . . . 284
initialize . . . 284
getSupportedProductNames . . . 284
getSupportedProductIDs . . . 285
isSupportedProductID . . . 285
print . . . 285
dac . . . 285
dio . . . 285
33.144.4 Friends And Related Function Documentation . . . 285
USBDeviceManager . . . 285
33.144.5 Field Documentation . . . 285
supportedProductIDs . . . 285
analogOutputSubsystem . . . 285
digitalIOSubsystem . . . 285
33.145 USB_CONFIGURATION_DESCRIPTOR Struct Reference . . . 286
33.145.1 Field Documentation . . . 286
bLength . . . 286
bDescriptorType . . . 286
wTotalLength . . . 286
bNumInterfaces . . . 286
bConfigurationValue . . . 286
iConfiguration . . . 286
bmAttributes . . . 286
MaxPower . . . 286
33.146 USB_CONFIGURATION_DESCRIPTOR_SHORT Struct Reference . . . 286
33.146.1 Field Documentation . . . 286
ConnectionIndex . . . 286
bmRequest . . . 286
bRequest . . . 286
wValue . . . 286
wIndex . . . 287
wLength . . . 287
SetupPacket . . . 287
req . . . 287
data . . . 287
33.147 USB_CTR_15_Family Class Reference . . . 287
33.147.1 Detailed Description . . . 288
33.147.2 Constructor & Destructor Documentation . . . 288
USB_CTR_15_Family . . . 288
~USB_CTR_15_Family . . . 288
33.147.3 Member Function Documentation . . . 288
initialize . . . 288
getSupportedProductNames . . . 288

getSupportedProductIDs . . . 288
isSupportedProductID . . . 288
print . . . 288
ctr . . . 289
33.147.4 Friends And Related Function Documentation . . . 289
USBDeviceManager . . . 289
33.147.5 Field Documentation . . . 289
supportedProductIDs . . . 289
counterSubsystem . . . 289
33.148 USB_DA12_8A_Family Class Reference . . . 289
33.148.1 Detailed Description . . . 290
33.148.2 Constructor & Destructor Documentation . . . 290
USB_DA12_8A_Family . . . 290
~USB_DA12_8A_Family . . . 290
33.148.3 Member Function Documentation . . . 290
initialize . . . 290
getSupportedProductNames . . . 290
getSupportedProductIDs . . . 290
isSupportedProductID . . . 290
print . . . 291
dac . . . 291
33.148.4 Friends And Related Function Documentation . . . 291
USBDeviceManager . . . 291
33.148.5 Field Documentation . . . 291
supportedProductIDs . . . 291
analogOutputSubsystem . . . 291
33.149 USB_DA12_8E_Family Class Reference . . . 291
33.149.1 Detailed Description . . . 292
33.149.2 Constructor & Destructor Documentation . . . 292
USB_DA12_8E_Family . . . 292
~USB_DA12_8E_Family . . . 292
33.149.3 Member Function Documentation . . . 292
initialize . . . 292
getSupportedProductNames . . . 292
getSupportedProductIDs . . . 292
isSupportedProductID . . . 293
print . . . 294
dac . . . 294
33.149.4 Friends And Related Function Documentation . . . 294
USBDeviceManager . . . 294
33.149.5 Field Documentation . . . 294
supportedProductIDs . . . 294
analogOutputSubsystem . . . 294
33.150 usb_descriptor_header Struct Reference . . . 294
33.150.1 Field Documentation . . . 294
bLength . . . 294
bDescriptorType . . . 294
33.151 USB_DESCRIPTOR_REQUEST Struct Reference . . . 295
33.151.1 Field Documentation . . . 295

ConnectionIndex . . . 295
bmRequest . . . 295
bRequest . . . 295
wValue . . . 295
wIndex . . . 295
wLength . . . 295
SetupPacket . . . 295
33.152 USB_DIO_16_Family Class Reference . . . 295
33.152.1 Detailed Description . . . 296
33.152.2 Constructor & Destructor Documentation . . . 296
USB_DIO_16_Family . . . 296
~USB_DIO_16_Family . . . 296
33.152.3 Member Function Documentation . . . 296
initialize . . . 296
getSupportedProductNames . . . 296
getSupportedProductIDs . . . 297
isSupportedProductID . . . 297
print . . . 297
dio . . . 297
diostream . . . 297
33.152.4 Friends And Related Function Documentation . . . 297
USBDeviceManager . . . 297
33.152.5 Field Documentation . . . 297
supportedProductIDs . . . 297
digitalIOSubsystem . . . 297
dioStreamSubsystem . . . 297
33.153 USB_DIO_32_Family Class Reference . . . 298
33.153.1 Detailed Description . . . 298
33.153.2 Constructor & Destructor Documentation . . . 299
USB_DIO_32_Family . . . 299
~USB_DIO_32_Family . . . 299
33.153.3 Member Function Documentation . . . 299
initialize . . . 299
getSupportedProductNames . . . 299
getSupportedProductIDs . . . 299
isSupportedProductID . . . 299
print . . . 299
dio . . . 299
ctr . . . 300
33.153.4 Friends And Related Function Documentation . . . 300
USBDeviceManager . . . 300
33.153.5 Field Documentation . . . 300
supportedProductIDs . . . 300
digitalIOSubsystem . . . 300
counterSubsystem . . . 300
33.154 USB_DIO_Family Class Reference . . . 300
33.154.1 Detailed Description . . . 301
33.154.2 Constructor & Destructor Documentation . . . 301
USB_DIO_Family . . . 301

~USB_DIO_Family . . . 301
33.154.3 Member Function Documentation . . . 301
initialize . . . 301
getSupportedProductNames . . . 301
getSupportedProductIDs . . . 301
isSupportedProductID . . . 301
print . . . 302
dio . . . 302
33.154.4 Friends And Related Function Documentation . . . 302
USBDeviceManager . . . 302
33.154.5 Field Documentation . . . 302
supportedProductIDs . . . 302
digitalIOSubsystem . . . 302
33.155 USB_ENDPOINT_DESCRIPTOR Struct Reference . . . 302
33.155.1 Field Documentation . . . 302
bLength . . . 302
bDescriptorType . . . 302
bEndpointAddress . . . 302
bmAttributes . . . 302
wMaxPacketSize . . . 302
bInterval . . . 302
33.156 USB_HUB_CAP_FLAGS Struct Reference . . . 303
33.156.1 Field Documentation . . . 303
HubIsHighSpeedCapable . . . 303
HubIsHighSpeed . . . 303
HubIsMultiTtCapable . . . 303
HubIsMultiTt . . . 303
HubIsRoot . . . 303
HubIsArmedWakeOnConnect . . . 303
ReservedMBZ . . . 303
33.157 USB_HUB_CAPABILITIES Struct Reference . . . 303
33.157.1 Field Documentation . . . 303
HubIs2xCapable . . . 303
33.158 USB_HUB_CAPABILITIES_EX Struct Reference . . . 303
33.158.1 Field Documentation . . . 304
CapabilityFlags . . . 304
33.159 USB_HUB_DESCRIPTOR Struct Reference . . . 304
33.159.1 Field Documentation . . . 304
bDescriptorLength . . . 304
bDescriptorType . . . 304
bNumberOfPorts . . . 304
wHubCharacteristics . . . 304
bPowerOnToPowerGood . . . 304
bHubControlCurrent . . . 304
bRemoveAndPowerMask . . . 304
33.160 USB_HUB_INFORMATION Struct Reference . . . 304
33.160.1 Field Documentation . . . 304
HubDescriptor . . . 304
HubIsBusPowered . . . 304

33.161 USB_HUB_NAME_FIXED Struct Reference . . . 305
33.161.1 Field Documentation . . . 305
root . . . 305
node . . . 305
u . . . 305
33.162 USB_INTERFACE_DESCRIPTOR Struct Reference . . . 305
33.162.1 Field Documentation . . . 305
bLength . . . 305
bDescriptorType . . . 305
bInterfaceNumber . . . 305
bAlternateSetting . . . 305
bNumEndpoints . . . 305
bInterfaceClass . . . 305
bInterfaceSubClass . . . 305
bInterfaceProtocol . . . 305
iInterface . . . 305
33.163 USB_MI_PARENT_INFORMATION Struct Reference . . . 306
33.163.1 Field Documentation . . . 306
NumberOfInterfaces . . . 306
33.164 USB_NODE_CONNECTION_INFORMATION_EX Struct Reference . . . 306
33.164.1 Field Documentation . . . 306
ConnectionIndex . . . 306
DeviceDescriptor . . . 306
CurrentConfigurationValue . . . 306
Speed . . . 306
DeviceIsHub . . . 306
DeviceAddress . . . 306
NumberOfOpenPipes . . . 306
ConnectionStatus . . . 306
33.165 USB_NODE_CONNECTION_NAME Struct Reference . . . 306
33.165.1 Field Documentation . . . 307
ConnectionIndex . . . 307
ActualLength . . . 307
NodeName . . . 307
33.166 USB_NODE_CONNECTION_NAME_FIXED Struct Reference . . . 307
33.166.1 Field Documentation . . . 307
ConnectionIndex . . . 307
ActualLength . . . 307
NodeName . . . 307
33.167 USB_NODE_INFORMATION Struct Reference . . . 307
33.167.1 Field Documentation . . . 307
NodeType . . . 307
HubInformation . . . 307
MiParentInformation . . . 307
u . . . 307
33.168 USB_PIPE_INFO Struct Reference . . . 307
33.168.1 Field Documentation . . . 308
EndpointDescriptor . . . 308
ScheduleOffset . . . 308

33.169usb_raw_command Union Reference . . . 308
33.169.1Field Documentation . . . 310
status . . . 310
version . . . 310
descriptor . . . 310
device . . . 310
descriptor . . . 310
config_index . . . 310
config . . . 310
alternate_info . . . 310
interface_index . . . 310
alternate . . . 310
descriptor . . . 310
interface . . . 310
alternate_index . . . 310
interface_etc . . . 310
descriptor . . . 310
endpoint_index . . . 310
endpoint . . . 310
endpoint_etc . . . 310
descriptor . . . 310
generic_index . . . 310
length . . . 310
generic . . . 310
generic_etc . . . 310
descriptor . . . 310
string_index . . . 310
string . . . 310
type . . . 310
index . . . 310
language_id . . . 310
data . . . 310
descriptor . . . 310
request_type . . . 310
request . . . 310
value . . . 310
index . . . 311
length . . . 311
control . . . 311
interface . . . 311
endpoint . . . 311
transfer . . . 311
packet_descriptors . . . 311
packet_count . . . 311
isochronous . . . 311
33.170USB_ROOT_HUB_NAME Struct Reference . . . 311
33.170.1Field Documentation . . . 311
ActualLength . . . 311
RootHubName . . . 311

33.171 USB_ROOT_HUB_NAME_FIXED Struct Reference . . . 311
33.171.1 Field Documentation . . . 311
ActualLength . . . 311
RootHubName . . . 311
33.172 USBDevice Struct Reference . . . 311
33.172.1 Constructor & Destructor Documentation . . . 313
USBDevice . . . 313
~USBDevice . . . 313
33.172.2 Member Function Documentation . . . 313
Location . . . 313
CountConfigurations . . . 313
Descriptor . . . 313
ConfigurationDescriptor . . . 313
ActiveConfiguration . . . 313
EndpointToIndex . . . 313
EndpointToInterface . . . 313
ClaimInterface . . . 313
ReleaseInterface . . . 313
CheckInterfacesFree . . . 313
SetActiveConfiguration . . . 313
ActiveConfigurationIndex . . . 313
InitCheck . . . 313
Initialise . . . 313
33.172.3 Field Documentation . . . 313
usb_control_transfer . . . 313
usb_bulk_transfer . . . 313
usb_request . . . 313
usb_reset_device . . . 313
usb_put_config . . . 313
usb_get_config . . . 313
timeout . . . 313
device . . . 313
deviceHandle . . . 313
deviceDesc . . . 313
debug . . . 313
usblp_attached . . . 313
iface . . . 313
verbose . . . 313
conf . . . 314
origconf . . . 314
altset . . . 314
fClaimedInterfaces . . . 314
fDeviceDescriptor . . . 314
fConfigurationDescriptors . . . 314
fActiveConfiguration . . . 314
fPath . . . 314
fConfigToIndex . . . 314
fEndpointToIndex . . . 314
fEndpointToInterface . . . 314

fInitCheck . . . 314
33.173 USBDeviceArray Class Reference . . . 314
33.173.1 Constructor & Destructor Documentation . . . 314
USBDeviceArray . . . 314
33.174 USBDeviceBase Class Reference . . . 314
33.174.1 Detailed Description . . . 315
33.174.2 Constructor & Destructor Documentation . . . 315
USBDeviceBase . . . 315
~USBDeviceBase . . . 315
33.174.3 Member Function Documentation . . . 315
clearFIFO . . . 315
getMiscClock . . . 315
setMiscClock . . . 316
getStreamingBlockSize . . . 316
setStreamingBlockSize . . . 316
print . . . 316
getDeviceIndex . . . 316
getProductID . . . 316
getName . . . 316
getSerialNumber . . . 316
getCommTimeout . . . 316
setCommTimeout . . . 317
reset . . . 317
customEEPROMWrite . . . 317
customEEPROMRead . . . 317
33.174.4 Friends And Related Function Documentation . . . 318
USBDeviceManager . . . 318
DIOStreamSubsystem . . . 318
AnalogInputSubsystem . . . 318
33.174.5 Field Documentation . . . 318
CUSTOM_EEPROM_SIZE . . . 318
CLEAR_FIFO_METHOD_IMMEDIATE . . . 318
CLEAR_FIFO_METHOD_AUTO . . . 318
CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT . . . 318
CLEAR_FIFO_METHOD_WAIT . . . 318
deviceIndex . . . 318
productID . . . 318
name . . . 318
serialNumber . . . 318
33.175 USBDeviceHandle Class Reference . . . 318
33.175.1 Constructor & Destructor Documentation . . . 319
USBDeviceHandle . . . 319
~USBDeviceHandle . . . 319
33.175.2 Member Function Documentation . . . 319
ClaimInterface . . . 319
ReleaseInterface . . . 319
SetConfiguration . . . 319
SetAltSetting . . . 319
SubmitTransfer . . . 319

CancelTransfer . . . 319
InitCheck . . . 319
TransfersThread . . . 319
TransfersWorker . . . 319
33.175.3 Field Documentation . . . 319
fRawFD . . . 319
fUSBDevice . . . 319
fClaimedInterfaces . . . 319
fTransfers . . . 319
fTransfersLock . . . 319
fTransfersSem . . . 319
fTransfersThread . . . 319
fInitCheck . . . 319
33.176 USBDeviceManager Class Reference . . . 320
33.176.1 Detailed Description . . . 321
33.176.2 Constructor & Destructor Documentation . . . 321
USBDeviceManager . . . 321
~USBDeviceManager . . . 321
33.176.3 Member Function Documentation . . . 321
emptyDeviceList . . . 321
print . . . 321
printDevices . . . 321
getAIOUSBVersion . . . 321
getAIOUSBVersionDate . . . 321
productIDToName . . . 322
productIDToName . . . 322
productNameToID . . . 322
productNameToID . . . 323
listDevices . . . 323
getResultCodeAsString . . . 323
isOpen . . . 323
open . . . 323
close . . . 324
scanForDevices . . . 324
getDeviceByProductID . . . 324
getDeviceByProductID . . . 324
getDeviceByProductID . . . 325
getDeviceBySerialNumber . . . 325
33.176.4 Field Documentation . . . 325
VERSION_NUMBER . . . 325
VERSION_DATE . . . 325
MIN_PRODUCT_ID . . . 325
MAX_PRODUCT_ID . . . 325
deviceList . . . 325
OPEN_PATTERN . . . 325
openStatus . . . 325
MESSAGE_NOT_OPEN . . . 325
33.177 usbfs_bulktransfer Struct Reference . . . 326
33.177.1 Field Documentation . . . 326

ep . . . 326
len . . . 326
timeout . . . 326
data . . . 326
33.178usbfs_connectinfo Struct Reference . . . 326
33.178.1Field Documentation . . . 326
devnum . . . 326
slow . . . 326
33.179usbfs_ctrltransfer Struct Reference . . . 326
33.179.1Field Documentation . . . 327
bmRequestType . . . 327
bRequest . . . 327
wValue . . . 327
wIndex . . . 327
wLength . . . 327
timeout . . . 327
data . . . 327
33.180usbfs_disconnect_claim Struct Reference . . . 327
33.180.1Field Documentation . . . 327
interface . . . 327
flags . . . 327
driver . . . 327
33.181usbfs_getdriver Struct Reference . . . 327
33.181.1Field Documentation . . . 327
interface . . . 327
driver . . . 327
33.182usbfs_hub_portinfo Struct Reference . . . 328
33.182.1Field Documentation . . . 328
numports . . . 328
port . . . 328
33.183usbfs_ioctl Struct Reference . . . 328
33.183.1Field Documentation . . . 328
ifno . . . 328
ioctl_code . . . 328
data . . . 328
33.184usbfs_iso_packet_desc Struct Reference . . . 328
33.184.1Field Documentation . . . 328
length . . . 328
actual_length . . . 328
status . . . 328
33.185usbfs_setinterface Struct Reference . . . 329
33.185.1Field Documentation . . . 329
interface . . . 329
altsetting . . . 329
33.186usbfs_streams Struct Reference . . . 329
33.186.1Field Documentation . . . 329
num_streams . . . 329
num_eps . . . 329
eps . . . 329

33.187usbfs_urb Struct Reference . . . 329
33.187.1Field Documentation . . . 330
type . . . 330
endpoint . . . 330
status . . . 330
flags . . . 330
buffer . . . 330
buffer_length . . . 330
actual_length . . . 330
start_frame . . . 330
number_of_packets . . . 330
stream_id . . . 330
"@34 . . . 330
error_count . . . 330
signr . . . 330
usercontext . . . 330
iso_frame_desc . . . 330
33.188usbi_cond Struct Reference . . . 330
33.188.1Field Documentation . . . 330
waiters . . . 330
not_waiting . . . 330
33.189usbi_cond_perthread Struct Reference . . . 330
33.189.1Field Documentation . . . 331
list . . . 331
tid . . . 331
event . . . 331
33.190usbi_os_backend Struct Reference . . . 331
33.190.1Field Documentation . . . 332
name . . . 332
caps . . . 332
init . . . 332
exit . . . 332
get_device_list . . . 332
hotplug_poll . . . 332
open . . . 332
close . . . 332
get_device_descriptor . . . 332
get_active_config_descriptor . . . 332
get_config_descriptor . . . 332
get_config_descriptor_by_value . . . 332
get_configuration . . . 332
set_configuration . . . 332
claim_interface . . . 332
release_interface . . . 332
set_interface_altsetting . . . 332
clear_halt . . . 332
reset_device . . . 332
alloc_streams . . . 332
free_streams . . . 332

dev_mem_alloc . . . 332
dev_mem_free . . . 332
kernel_driver_active . . . 332
detach_kernel_driver . . . 332
attach_kernel_driver . . . 332
destroy_device . . . 332
submit_transfer . . . 332
cancel_transfer . . . 332
clear_transfer_priv . . . 332
handle_events . . . 332
handle_transfer_completion . . . 332
clock_gettime . . . 333
device_priv_size . . . 333
device_handle_priv_size . . . 333
transfer_priv_size . . . 333
33.191usbi_pollfd Struct Reference . . . 333
33.191.1Field Documentation . . . 333
pollfd . . . 333
list . . . 333
33.192usbi_transfer Struct Reference . . . 333
33.192.1Field Documentation . . . 333
num_iso_packets . . . 333
list . . . 333
completed_list . . . 333
timeout . . . 333
transferred . . . 333
stream_id . . . 333
state_flags . . . 333
timeout_flags . . . 334
lock . . . 334
33.193USBRoster Class Reference . . . 334
33.193.1Constructor & Destructor Documentation . . . 334
USBRoster . . . 334
∼USBRoster . . . 334
33.193.2Member Function Documentation . . . 334
Start . . . 334
Stop . . . 334
33.193.3Field Documentation . . . 334
fLooper . . . 334
33.194USBTransfer Class Reference . . . 334
33.194.1Constructor & Destructor Documentation . . . 335
USBTransfer . . . 335
∼USBTransfer . . . 335
33.194.2Member Function Documentation . . . 335
Do . . . 335
UsbiTransfer . . . 335
SetCancelled . . . 335
IsCancelled . . . 335
33.194.3Field Documentation . . . 335

fUsbiTransfer . . . 335
fLibusbTransfer . . . 335
fUSBDevice . . . 335
fStatusLock . . . 335
fCancelled . . . 335
33.195 fshort_array Struct Reference . . . 335
33.195.1 Field Documentation . . . 335
size . . . 335
33.196 UShortArray Class Reference . . . 335
33.196.1 Constructor & Destructor Documentation . . . 335
UShortArray . . . 335
33.197 WatchedEntry Class Reference . . . 336
33.197.1 Constructor & Destructor Documentation . . . 336
WatchedEntry . . . 336
~WatchedEntry . . . 336
33.197.2 Member Function Documentation . . . 336
EntryCreated . . . 336
EntryRemoved . . . 336
InitCheck . . . 336
33.197.3 Field Documentation . . . 336
fMessenger . . . 336
fNode . . . 336
fIsDirectory . . . 336
fDevice . . . 336
fEntries . . . 336
fLink . . . 336
fInitCheck . . . 336
33.198 wince_device_priv Struct Reference . . . 336
33.198.1 Field Documentation . . . 337
dev . . . 337
desc . . . 337
33.199 wince_transfer_priv Struct Reference . . . 337
33.199.1 Field Documentation . . . 337
pollable_fd . . . 337
interface_number . . . 337
33.200 windows_device_handle_priv Struct Reference . . . 337
33.200.1 Field Documentation . . . 337
active_interface . . . 337
interface_handle . . . 337
autoclaim_count . . . 337
33.201 windows_device_priv Struct Reference . . . 337
33.201.1 Field Documentation . . . 338
depth . . . 338
port . . . 338
active_config . . . 338
parent_dev . . . 338
apib . . . 338
path . . . 338
sub_api . . . 338

nb_endpoints . . . 338
endpoint . . . 338
restricted_functionality . . . 338
usb_interface . . . 338
hid . . . 338
dev_descriptor . . . 338
config_descriptor . . . 338
33.202 windows_transfer_priv Struct Reference . . . 338
33.202.1 Field Documentation . . . 339
pollable_fd . . . 339
interface_number . . . 339
hid_buffer . . . 339
hid_dest . . . 339
hid_expected_size . . . 339
33.203 windows_usb_api_backend Struct Reference . . . 339
33.203.1 Field Documentation . . . 339
id . . . 339
designation . . . 339
driver_name_list . . . 339
nb_driver_names . . . 339
init . . . 339
exit . . . 339
open . . . 339
close . . . 339
configure_endpoints . . . 339
claim_interface . . . 339
set_interface_altsetting . . . 339
release_interface . . . 340
clear_halt . . . 340
reset_device . . . 340
submit_bulk_transfer . . . 340
submit_iso_transfer . . . 340
submit_control_transfer . . . 340
abort_control . . . 340
abort_transfers . . . 340
copy_transfer_data . . . 340
33.204 winfd Struct Reference . . . 340
33.204.1 Field Documentation . . . 340
fd . . . 340
handle . . . 340
overlapped . . . 340
itransfer . . . 340
cancel_fn . . . 340
rw . . . 340
33.205 winusb_interface Struct Reference . . . 340
33.205.1 Field Documentation . . . 341
initialized . . . 341
AbortPipe . . . 341
ControlTransfer . . . 341

FlushPipe . . . 341
Free . . . 341
GetAssociatedInterface . . . 341
GetCurrentAlternateSetting . . . 341
GetDescriptor . . . 341
GetOverlappedResult . . . 341
GetPipePolicy . . . 341
GetPowerPolicy . . . 341
Initialize . . . 341
QueryDeviceInformation . . . 341
QueryInterfaceSettings . . . 341
QueryPipe . . . 341
ReadPipe . . . 341
ResetPipe . . . 341
SetCurrentAlternateSetting . . . 341
SetPipePolicy . . . 341
SetPowerPolicy . . . 341
WritePipe . . . 341
ResetDevice . . . 341
33.206 WINUSB_PIPE_INFORMATION Struct Reference . . . 342
33.206.1 Field Documentation . . . 342
PipeType . . . 342
PipeId . . . 342
MaximumPacketSize . . . 342
Interval . . . 342
33.207 WINUSB_SETUP_PACKET Struct Reference . . . 342
33.207.1 Field Documentation . . . 342
request_type . . . 342
request . . . 342
value . . . 342
index . . . 342
length . . . 342

34 File Documentation **343**
34.1 deprecated/classlib/AI16_DataPoint.cpp File Reference . . . 343
34.2 deprecated/classlib/AI16_DataPoint.hpp File Reference . . . 343
34.3 deprecated/classlib/AI16_DataSet.cpp File Reference . . . 343
34.4 deprecated/classlib/AI16_DataSet.hpp File Reference . . . 343
34.5 deprecated/classlib/AI16_InputRange.cpp File Reference . . . 344
34.6 deprecated/classlib/AI16_InputRange.hpp File Reference . . . 344
34.7 deprecated/classlib/AnalogInputSubsystem.cpp File Reference . . . 344
34.7.1 Detailed Description . . . 344
34.8 deprecated/classlib/AnalogInputSubsystem.hpp File Reference . . . 345
34.9 deprecated/classlib/AnalogIORange.cpp File Reference . . . 345
34.10 deprecated/classlib/AnalogIORange.hpp File Reference . . . 345
34.11 deprecated/classlib/AnalogOutputSubsystem.cpp File Reference . . . 345
34.12 deprecated/classlib/AnalogOutputSubsystem.hpp File Reference . . . 346
34.13 deprecated/classlib/AO16_AnalogOutputSubsystem.cpp File Reference . . . 346
34.14 deprecated/classlib/AO16_AnalogOutputSubsystem.hpp File Reference . . . 346
34.15 deprecated/classlib/AO16_OutputRange.cpp File Reference . . . 346

34.16deprecated/classlib/AO16_OutputRange.hpp File Reference . . . 346
34.17deprecated/classlib/Counter.cpp File Reference . . . 347
34.17.1 Detailed Description . . . 347
34.18deprecated/classlib/Counter.hpp File Reference . . . 347
34.19deprecated/classlib/CounterSubsystem.cpp File Reference . . . 348
34.19.1 Detailed Description . . . 348
34.20deprecated/classlib/CounterSubsystem.hpp File Reference . . . 348
34.21deprecated/classlib/CppCommon.h File Reference . . . 348
34.22deprecated/classlib/DA12_AnalogOutputSubsystem.cpp File Reference . . . 348
34.22.1 Detailed Description . . . 349
34.23deprecated/classlib/DA12_AnalogOutputSubsystem.hpp File Reference . . . 349
34.24deprecated/classlib/DA12_OutputRange.cpp File Reference . . . 349
34.25deprecated/classlib/DA12_OutputRange.hpp File Reference . . . 349
34.26deprecated/classlib/DeviceSubsystem.cpp File Reference . . . 350
34.27deprecated/classlib/DeviceSubsystem.hpp File Reference . . . 350
34.28deprecated/classlib/DigitalIOSubsystem.cpp File Reference . . . 350
34.28.1 Detailed Description . . . 350
34.29deprecated/classlib/DigitalIOSubsystem.hpp File Reference . . . 350
34.30deprecated/classlib/DIOStreamSubsystem.cpp File Reference . . . 351
34.30.1 Detailed Description . . . 351
34.31deprecated/classlib/DIOStreamSubsystem.hpp File Reference . . . 351
34.32deprecated/classlib/OutputVoltagePoint.hpp File Reference . . . 351
34.33deprecated/classlib/README.doc File Reference . . . 352
34.34lib/wrappers/README.doc File Reference . . . 352
34.35Firmware/README.doc File Reference . . . 352
34.36samples/USB-AI16-16/README.doc File Reference . . . 352
34.37samples/USB-AO16-16/README.doc File Reference . . . 352
34.38samples/USB-DA12-8A/README.doc File Reference . . . 352
34.39samples/USB-DIO-16/README.doc File Reference . . . 352
34.40samples/USB-DIO-32/README.doc File Reference . . . 352
34.41samples/USB-IDIO-16_8/README.doc File Reference . . . 352
34.42samples/USB-IIRO-16_8/README.doc File Reference . . . 352
34.43deprecated/classlib/USB_AI16_Family.cpp File Reference . . . 352
34.44deprecated/classlib/USB_AI16_Family.hpp File Reference . . . 352
34.45deprecated/classlib/USB_AIO16_Family.cpp File Reference . . . 353
34.46deprecated/classlib/USB_AIO16_Family.hpp File Reference . . . 353
34.47deprecated/classlib/USB_AO16_Family.cpp File Reference . . . 353
34.48deprecated/classlib/USB_AO16_Family.hpp File Reference . . . 353
34.49deprecated/classlib/USB_CTR_15_Family.cpp File Reference . . . 354
34.50deprecated/classlib/USB_CTR_15_Family.hpp File Reference . . . 354
34.51deprecated/classlib/USB_DA12_8A_Family.cpp File Reference . . . 354
34.52deprecated/classlib/USB_DA12_8A_Family.hpp File Reference . . . 354
34.53deprecated/classlib/USB_DA12_8E_Family.cpp File Reference . . . 355
34.54deprecated/classlib/USB_DA12_8E_Family.hpp File Reference . . . 355
34.55deprecated/classlib/USB_DIO_16_Family.cpp File Reference . . . 355
34.56deprecated/classlib/USB_DIO_16_Family.hpp File Reference . . . 355
34.57deprecated/classlib/USB_DIO_32_Family.cpp File Reference . . . 356
34.58deprecated/classlib/USB_DIO_32_Family.hpp File Reference . . . 356
34.59deprecated/classlib/USB_DIO_Family.cpp File Reference . . . 356

34.60 deprecated/classlib/USB_DIO_Family.hpp File Reference . . . 356
34.61 deprecated/classlib/USBDeviceBase.cpp File Reference . . . 357
34.62 deprecated/classlib/USBDeviceBase.hpp File Reference . . . 357
34.63 deprecated/classlib/USBDeviceManager.cpp File Reference . . . 358
34.63.1 Detailed Description . . . 358
34.64 deprecated/classlib/USBDeviceManager.hpp File Reference . . . 358
34.64.1 Detailed Description . . . 359
34.65 doc/aiousb.doc File Reference . . . 359
34.66 doc/firmware.doc File Reference . . . 359
34.67 doc/index.doc File Reference . . . 359
34.68 doc/install.doc File Reference . . . 359
34.69 doc/java.doc File Reference . . . 359
34.70 doc/libusb.doc File Reference . . . 359
34.71 doc/samples.doc File Reference . . . 359
34.72 doc/wrappers.doc File Reference . . . 359
34.73 lib/ADCConfigBlock.c File Reference . . . 359
34.73.1 Function Documentation . . . 360
ADCConfigBlockCopy . . . 360
DeleteADCConfigBlock . . . 360
ADCConfigBlockGetAIOUSBDevice . . . 360
ADCConfigBlockSetAIOUSBDevice . . . 360
ADCConfigBlockSetDevice . . . 360
ADCConfigBlockInitializeDefault . . . 360
ADCConfigBlockInitializeFromAIOUSBDevice . . . 360
ADCConfigBlockSetSize . . . 361
ADCConfigBlockGetSize . . . 361
ADCConfigBlockSetTesting . . . 361
ADCConfigBlockSetDebug . . . 361
ADCConfigBlockSetRangeSingle . . . 361
ADCConfigBlockSetRegister . . . 361
ADCConfigBlockGetTesting . . . 361
ADCConfigBlockGetDebug . . . 361
ADCConfigBlockInit . . . 361
ADCConfigBlockInitForCounterScan . . . 361
ADC_VerifyAndCorrectConfigBlock . . . 361
ADCConfigBlockSetAllGainCodeAndDiffMode . . . 361
ADCConfigBlockGetGainCode . . . 361
ADCConfigBlockSetGainCode . . . 361
ADCConfigBlockSetEndChannel . . . 361
ADCConfigBlockSetChannelRange . . . 361
ADCConfigBlockSetStartChannel . . . 361
ADCConfigBlockSetScanRange . . . 361
ADCConfigBlockSetCalMode . . . 361
ADCConfigBlockGetCalMode . . . 361
ADCConfigBlockGetStartChannel . . . 362
ADCConfigBlockGetEndChannel . . . 362
ADCConfigBlockGetOversample . . . 362
ADCConfigBlockSetOversample . . . 362
ADCConfigBlockGetTimeout . . . 362

ADCConfigBlockSetTimeout . . . 362
ADCConfigBlockGetTriggerMode . . . 362
ADCConfigBlockSetTriggerMode . . . 362
ADCConfigBlockSetDifferentialMode . . . 362
ADCConfigBlockSetReference . . . 362
ADCConfigBlockSetTriggerEdge . . . 362
get_gain_code . . . 362
get_cal_mode . . . 362
ADCConfigBlockToYAML . . . 362
ADCConfigBlockToJSON . . . 362
ADCConfigBlockSetScanAllChannels . . . 362
ADCConfigBlockSetTriggerReference . . . 362
is_all_digits . . . 362
ADCConfigBlockGetJSONValueOrDefault . . . 362
ADCConfigBlockGetJSONValueOrInt . . . 363
NewADCConfigBlockFromJSON . . . 363
ADCConfigBlockSetClockRate . . . 363
ADCConfigBlockGetClockRate . . . 363
34.74 lib/ndkbuild/jni/ADCConfigBlock.c File Reference . . . 363
34.74.1 Function Documentation . . . 364
ADCConfigBlockCopy . . . 364
DeleteADCConfigBlock . . . 364
ADCConfigBlockGetAIOUSBDevice . . . 364
ADCConfigBlockSetAIOUSBDevice . . . 364
ADCConfigBlockSetDevice . . . 364
ADCConfigBlockInitializeDefault . . . 364
ADCConfigBlockInitializeFromAIOUSBDevice . . . 364
ADCConfigBlockSetSize . . . 364
ADCConfigBlockGetSize . . . 364
ADCConfigBlockSetTesting . . . 364
ADCConfigBlockSetDebug . . . 364
ADCConfigBlockSetRangeSingle . . . 364
ADCConfigBlockSetRegister . . . 364
ADCConfigBlockGetTesting . . . 364
ADCConfigBlockGetDebug . . . 364
ADCConfigBlockInit . . . 364
ADCConfigBlockInitForCounterScan . . . 364
ADC_VerifyAndCorrectConfigBlock . . . 365
ADCConfigBlockSetAllGainCodeAndDiffMode . . . 365
ADCConfigBlockGetGainCode . . . 365
ADCConfigBlockSetGainCode . . . 365
ADCConfigBlockSetEndChannel . . . 365
ADCConfigBlockSetChannelRange . . . 365
ADCConfigBlockSetStartChannel . . . 365
ADCConfigBlockSetScanRange . . . 365
ADCConfigBlockSetCalMode . . . 365
ADCConfigBlockGetCalMode . . . 365
ADCConfigBlockGetStartChannel . . . 365
ADCConfigBlockGetEndChannel . . . 365

ADCConfigBlockGetOversample . . . 365
ADCConfigBlockSetOversample . . . 365
ADCConfigBlockGetTimeout . . . 365
ADCConfigBlockSetTimeout . . . 365
ADCConfigBlockGetTriggerMode . . . 365
ADCConfigBlockSetTriggerMode . . . 365
ADCConfigBlockSetDifferentialMode . . . 365
ADCConfigBlockSetReference . . . 365
ADCConfigBlockSetTriggerEdge . . . 365
get_gain_code . . . 365
get_cal_mode . . . 365
ADCConfigBlockToYAML . . . 366
ADCConfigBlockToJSON . . . 366
ADCConfigBlockSetScanAllChannels . . . 366
ADCConfigBlockSetTriggerReference . . . 366
is_all_digits . . . 366
ADCConfigBlockGetJSONValueOrDefault . . . 366
ADCConfigBlockGetJSONValueOrInt . . . 366
NewADCConfigBlockFromJSON . . . 366
ADCConfigBlockSetClockRate . . . 366
ADCConfigBlockGetClockRate . . . 366
34.75lib/ADCConfigBlock.h File Reference . . . 366
34.75.1 Typedef Documentation . . . 367
AIOUSBDevice . . . 367
ADCMuxSettings . . . 367
ADCConfigBlock . . . 367
ADConfigBlock . . . 367
34.75.2 Function Documentation . . . 368
ADCConfigBlockInit . . . 368
ADCConfigBlockInitForCounterScan . . . 369
ADCConfigBlockInitializeDefault . . . 369
ADC_VerifyAndCorrectConfigBlock . . . 369
ADCConfigBlockSetAllGainCodeAndDiffMode . . . 369
ADCConfigBlockSetRegister . . . 369
ADCConfigBlockGetGainCode . . . 369
ADCConfigBlockSetGainCode . . . 369
ADCConfigBlockSetClockRate . . . 369
ADCConfigBlockGetClockRate . . . 369
ADCConfigBlockSetScanRange . . . 369
ADCConfigBlockSetStartChannel . . . 369
ADCConfigBlockSetEndChannel . . . 369
ADCConfigBlockSetChannelRange . . . 369
ADCConfigBlockSetCalMode . . . 369
ADCConfigBlockGetCalMode . . . 369
ADCConfigBlockToYAML . . . 369
ADCConfigBlockGetStartChannel . . . 370
ADCConfigBlockGetEndChannel . . . 370
ADCConfigBlockGetOversample . . . 370
ADCConfigBlockSetOversample . . . 370

ADCConfigBlockGetTimeout . . . 370
ADCConfigBlockSetTimeout . . . 370
ADCConfigBlockGetTriggerMode . . . 370
ADCConfigBlockSetTriggerMode . . . 370
ADCConfigBlockSetReference . . . 370
ADCConfigBlockSetTriggerEdge . . . 370
ADCConfigBlockSetDifferentialMode . . . 370
ADCConfigBlockSetRangeSingle . . . 370
ADCConfigBlockCopy . . . 370
ADCConfigBlockSetDevice . . . 370
ADCConfigBlockSetAIOUSBDevice . . . 370
ADCConfigBlockGetAIOUSBDevice . . . 370
ADCConfigBlockInitializeFromAIOUSBDevice . . . 370
ADCConfigBlockSetTesting . . . 370
ADCConfigBlockGetTesting . . . 370
ADCConfigBlockSetSize . . . 370
ADCConfigBlockGetSize . . . 370
ADCConfigBlockSetDebug . . . 370
ADCConfigBlockGetDebug . . . 370
ADCConfigBlockToJSON . . . 371
NewADCConfigBlockFromJSON . . . 371
DeleteADCConfigBlock . . . 371
is_all_digits . . . 371
34.76lib/ndkbuild/jni/ADCConfigBlock.h File Reference . . . 371
34.76.1 Typedef Documentation . . . 372
AIOUSBDevice . . . 372
ADCMuxSettings . . . 372
ADCConfigBlock . . . 372
ADConfigBlock . . . 372
34.76.2 Function Documentation . . . 372
ADCConfigBlockInit . . . 372
ADCConfigBlockInitForCounterScan . . . 372
ADCConfigBlockInitializeDefault . . . 372
ADC_VerifyAndCorrectConfigBlock . . . 372
ADCConfigBlockSetAllGainCodeAndDiffMode . . . 372
ADCConfigBlockSetRegister . . . 372
ADCConfigBlockGetGainCode . . . 372
ADCConfigBlockSetGainCode . . . 372
ADCConfigBlockSetClockRate . . . 373
ADCConfigBlockGetClockRate . . . 373
ADCConfigBlockSetScanRange . . . 373
ADCConfigBlockSetStartChannel . . . 373
ADCConfigBlockSetEndChannel . . . 373
ADCConfigBlockSetChannelRange . . . 373
ADCConfigBlockSetCalMode . . . 373
ADCConfigBlockGetCalMode . . . 373
ADCConfigBlockToYAML . . . 373
ADCConfigBlockGetStartChannel . . . 373
ADCConfigBlockGetEndChannel . . . 373

ADCConfigBlockGetOversample . . . 373
ADCConfigBlockSetOversample . . . 373
ADCConfigBlockGetTimeout . . . 373
ADCConfigBlockSetTimeout . . . 373
ADCConfigBlockGetTriggerMode . . . 373
ADCConfigBlockSetTriggerMode . . . 373
ADCConfigBlockSetReference . . . 374
ADCConfigBlockSetTriggerEdge . . . 374
ADCConfigBlockSetDifferentialMode . . . 374
ADCConfigBlockSetRangeSingle . . . 374
ADCConfigBlockCopy . . . 374
ADCConfigBlockSetDevice . . . 374
ADCConfigBlockSetAIOUSBDevice . . . 374
ADCConfigBlockGetAIOUSBDevice . . . 374
ADCConfigBlockInitializeFromAIOUSBDevice . . . 374
ADCConfigBlockSetTesting . . . 374
ADCConfigBlockGetTesting . . . 374
ADCConfigBlockSetSize . . . 374
ADCConfigBlockGetSize . . . 374
ADCConfigBlockSetDebug . . . 374
ADCConfigBlockGetDebug . . . 374
ADCConfigBlockToJSON . . . 374
NewADCConfigBlockFromJSON . . . 374
DeleteADCConfigBlock . . . 374
is_all_digits . . . 374
34.77 lib/AIOBuf.c File Reference . . . 374
34.77.1 Function Documentation . . . 375
NewAIOBuf . . . 375
DeleteAIOBuf . . . 375
AIOBufGetSize . . . 375
AIOBufGetTotalSize . . . 375
AIOBufGetType . . . 375
AIOBufGetRaw . . . 375
AIOBufGetTypeSize . . . 375
AIOBufRead . . . 375
AIOBufWrite . . . 375
AIOBufGetIterator . . . 375
AIOBufIteratorIsValid . . . 375
AIOBufIteratorNext . . . 375
AIOBufIteratorGetValue . . . 375
34.78 lib/ndkbuild/jni/AIOBuf.c File Reference . . . 375
34.78.1 Function Documentation . . . 376
NewAIOBuf . . . 376
DeleteAIOBuf . . . 376
AIOBufGetSize . . . 376
AIOBufGetTotalSize . . . 376
AIOBufGetType . . . 376
AIOBufGetRaw . . . 376
AIOBufGetTypeSize . . . 376

AIOBufRead . . . 376
AIOBufWrite . . . 376
AIOBufGetIterator . . . 376
AIOBufIteratorIsValid . . . 376
AIOBufIteratorNext . . . 376
AIOBufIteratorGetValue . . . 376
34.79 lib/AIOBuf.h File Reference . . . 376
34.79.1 Typedef Documentation . . . 377
AIOBuf . . . 377
AIOBufIterator . . . 377
34.79.2 Enumeration Type Documentation . . . 377
AIOBufType . . . 377
34.79.3 Function Documentation . . . 377
NewAIOBuf . . . 377
DeleteAIOBuf . . . 377
AIOBufGetSize . . . 377
AIOBufGetType . . . 377
AIOBufGetRaw . . . 377
AIOBufRead . . . 377
AIOBufWrite . . . 377
AIOBufGetIterator . . . 377
AIOBufIteratorGetValue . . . 377
AIOBufIteratorIsValid . . . 378
AIOBufIteratorNext . . . 378
34.80 lib/ndkbuild/jni/AIOBuf.h File Reference . . . 378
34.80.1 Typedef Documentation . . . 379
AIOBuf . . . 379
AIOBufIterator . . . 379
34.80.2 Enumeration Type Documentation . . . 379
AIOBufType . . . 379
34.80.3 Function Documentation . . . 379
NewAIOBuf . . . 379
DeleteAIOBuf . . . 379
AIOBufGetSize . . . 379
AIOBufGetType . . . 379
AIOBufGetRaw . . . 379
AIOBufRead . . . 379
AIOBufWrite . . . 379
AIOBufGetIterator . . . 379
AIOBufIteratorGetValue . . . 379
AIOBufIteratorIsValid . . . 380
AIOBufIteratorNext . . . 380
34.81 lib/AIOChannelMask.c File Reference . . . 380
34.81.1 Function Documentation . . . 380
NewAIOChannelMask . . . 380
DeleteAIOChannelMask . . . 380
AIOChannelMaskIndices . . . 381
AIOChannelMaskNextIndex . . . 381
AIOChannelMaskSetMaskFromInt . . . 381

AIOChannelMaskSetMaskAtIndex . . . 381
AIOChannelMaskGetMaskAtIndex . . . 381
AIOChannelMaskGetSize . . . 381
AIOChannelMaskNumberChannels . . . 381
AIOChannelMaskNumberSignals . . . 381
AIOChannelMaskSetMaskFromStr . . . 381
NewAIOChannelMaskFromStr . . . 381
NewAIOChannelMaskFromChr . . . 382
AIOChannelMaskToString . . . 382
AIOChannelMaskToStringAtIndex . . . 382
AIOChannelMaskGetMask . . . 382
34.82 lib/ndkbuild/jni/AIOChannelMask.c File Reference . . . 382
34.82.1 Function Documentation . . . 383
NewAIOChannelMask . . . 383
DeleteAIOChannelMask . . . 383
AIOChannelMaskIndices . . . 383
AIOChannelMaskNextIndex . . . 383
AIOChannelMaskSetMaskFromInt . . . 383
AIOChannelMaskSetMaskAtIndex . . . 383
AIOChannelMaskGetMaskAtIndex . . . 383
AIOChannelMaskGetSize . . . 383
AIOChannelMaskNumberChannels . . . 383
AIOChannelMaskNumberSignals . . . 384
AIOChannelMaskSetMaskFromStr . . . 384
NewAIOChannelMaskFromStr . . . 384
NewAIOChannelMaskFromChr . . . 384
AIOChannelMaskToString . . . 384
AIOChannelMaskToStringAtIndex . . . 384
AIOChannelMaskGetMask . . . 384
34.83 lib/AIOChannelMask.h File Reference . . . 384
34.83.1 Macro Definition Documentation . . . 385
BIT_LENGTH . . . 385
34.83.2 Typedef Documentation . . . 385
aio_channel_obj . . . 385
34.83.3 Function Documentation . . . 385
NewAIOChannelMask . . . 385
DeleteAIOChannelMask . . . 385
NewAIOChannelMaskFromStr . . . 386
NewAIOChannelMaskFromChr . . . 386
AIOChannelMaskToString . . . 386
AIOChannelMaskToStringAtIndex . . . 386
AIOChannelMaskGetMask . . . 386
AIOChannelMaskGetMaskAtIndex . . . 386
AIOChannelMaskNumberChannels . . . 386
AIOChannelMaskNumberSignals . . . 387
AIOChannelMaskGetSize . . . 387
AIOChannelMaskIndices . . . 387
AIOChannelMaskNextIndex . . . 387
AIOChannelMaskSetMaskFromInt . . . 387

AIOChannelMaskSetMaskAtIndex . . . 387
AIOChannelMaskSetMaskFromStr . . . 387
34.84 lib/ndkbuild/jni/AIOChannelMask.h File Reference . . . 387
34.84.1 Macro Definition Documentation . . . 388
BIT_LENGTH . . . 388
34.84.2 Typedef Documentation . . . 388
aio_channel_obj . . . 388
34.84.3 Function Documentation . . . 388
NewAIOChannelMask . . . 388
DeleteAIOChannelMask . . . 388
NewAIOChannelMaskFromStr . . . 388
NewAIOChannelMaskFromChr . . . 389
AIOChannelMaskToString . . . 389
AIOChannelMaskToStringAtIndex . . . 389
AIOChannelMaskGetMask . . . 389
AIOChannelMaskGetMaskAtIndex . . . 389
AIOChannelMaskNumberChannels . . . 389
AIOChannelMaskNumberSignals . . . 390
AIOChannelMaskGetSize . . . 390
AIOChannelMaskIndices . . . 390
AIOChannelMaskNextIndex . . . 390
AIOChannelMaskSetMaskFromInt . . . 390
AIOChannelMaskSetMaskAtIndex . . . 390
AIOChannelMaskSetMaskFromStr . . . 390
34.85 lib/AIOChannelRange.c File Reference . . . 390
34.85.1 Enumeration Type Documentation . . . 391
STATE . . . 391
34.85.2 Function Documentation . . . 391
NewAIOChannelRangeFromStr . . . 391
DeleteAIOChannelRange . . . 391
lookup_voltage_range . . . 391
AIOChannelRangeToStr . . . 391
AIOChannelRangeGetStart . . . 391
AIOChannelRangeGetEnd . . . 391
AIOChannelRangeGetGain . . . 391
34.85.3 Variable Documentation . . . 391
aio_channel_range_error . . . 391
34.86 lib/ndkbuild/jni/AIOChannelRange.c File Reference . . . 391
34.86.1 Enumeration Type Documentation . . . 392
STATE . . . 392
34.86.2 Function Documentation . . . 392
NewAIOChannelRangeFromStr . . . 392
DeleteAIOChannelRange . . . 392
lookup_voltage_range . . . 392
AIOChannelRangeToStr . . . 392
AIOChannelRangeGetStart . . . 392
AIOChannelRangeGetEnd . . . 392
AIOChannelRangeGetGain . . . 392
34.86.3 Variable Documentation . . . 392

aio_channel_range_error . . . 392
34.87 lib/AIOChannelRange.h File Reference . . . 392
34.87.1 Macro Definition Documentation . . . 393
LENGTH_AD_GAIN_CODE_STRINGS . . . 393
34.87.2 Typedef Documentation . . . 393
AIOChannelRange . . . 393
34.87.3 Function Documentation . . . 393
NewAIOChannelRangeFromStr . . . 393
DeleteAIOChannelRange . . . 393
AIOChannelRangeToStr . . . 393
AIOChannelRangeGetStart . . . 393
AIOChannelRangeGetEnd . . . 393
AIOChannelRangeGetGain . . . 393
34.87.4 Variable Documentation . . . 393
AD_GAIN_CODE_STRINGS . . . 393
aio_channel_range_error . . . 393
34.88 lib/ndkbuild/jni/AIOChannelRange.h File Reference . . . 393
34.88.1 Macro Definition Documentation . . . 394
LENGTH_AD_GAIN_CODE_STRINGS . . . 394
34.88.2 Typedef Documentation . . . 394
AIOChannelRange . . . 394
34.88.3 Function Documentation . . . 394
NewAIOChannelRangeFromStr . . . 394
DeleteAIOChannelRange . . . 394
AIOChannelRangeToStr . . . 394
AIOChannelRangeGetStart . . . 394
AIOChannelRangeGetEnd . . . 394
AIOChannelRangeGetGain . . . 394
34.88.4 Variable Documentation . . . 394
AD_GAIN_CODE_STRINGS . . . 394
aio_channel_range_error . . . 394
34.89 lib/AIOCmd.c File Reference . . . 395
34.89.1 Function Documentation . . . 395
NewAIOCmdFromJSON . . . 395
NewAIOCmd . . . 395
DeleteAIOCmd . . . 395
34.90 lib/ndkbuild/jni/AIOCmd.c File Reference . . . 395
34.90.1 Function Documentation . . . 395
NewAIOCmdFromJSON . . . 395
NewAIOCmd . . . 395
DeleteAIOCmd . . . 395
34.91 lib/AIOCmd.h File Reference . . . 395
34.91.1 Typedef Documentation . . . 396
AIOCmd . . . 396
34.91.2 Function Documentation . . . 396
NewAIOCmdFromJSON . . . 396
NewAIOCmd . . . 396
DeleteAIOCmd . . . 396
34.92 lib/ndkbuild/jni/AIOCmd.h File Reference . . . 396

34.92.1 Typedef Documentation . . . 396
AIOCmd . . . 396
34.92.2 Function Documentation . . . 396
NewAIOCmdFromJSON . . . 396
NewAIOCmd . . . 396
DeleteAIOCmd . . . 396
34.93 lib/AIOCommandLine.c File Reference . . . 396
34.93.1 Function Documentation . . . 397
AIO_CMDLINE_DEFAULT_OPTIONS . . . 397
AIO_CMDLINE_SCRIPTING_OPTIONS . . . 397
AIO_CMDLINE_CLEAR_OPTIONS . . . 397
AIOProcessCommandLine . . . 397
AIOProcessCmdline . . . 397
AIOPrintUsage . . . 397
NewDefaultAIOCommandLineOptions . . . 398
NewAIOCommandLineOptionsFromDefaultOptions . . . 398
DeleteAIOCommandLineOptions . . . 398
AIOCommandLineListDevices . . . 398
AIOCommandLineOptionsListDevices . . . 398
AIOCommandLineOverrideADCConfigBlock . . . 398
AIOCommandLineOptionsOverrideADCConfigBlock . . . 398
AIOGetChannelRange . . . 398
AIO_SCRIPTING_OPTIONS . . . 398
AIO_CMDLINE_OPTIONS . . . 399
34.93.2 Variable Documentation . . . 399
opterr . . . 399
optind . . . 399
AIO_DEFAULT_CMDLINE_OPTIONS . . . 399
AIO_DEFAULT_SCRIPTING_OPTIONS . . . 399
34.94 lib/ndkbuild/jni/AIOCommandLine.c File Reference . . . 399
34.94.1 Function Documentation . . . 400
AIO_CMDLINE_DEFAULT_OPTIONS . . . 400
AIO_CMDLINE_SCRIPTING_OPTIONS . . . 400
AIO_CMDLINE_CLEAR_OPTIONS . . . 400
AIOProcessCommandLine . . . 400
AIOProcessCmdline . . . 400
AIOPrintUsage . . . 400
NewDefaultAIOCommandLineOptions . . . 400
NewAIOCommandLineOptionsFromDefaultOptions . . . 400
DeleteAIOCommandLineOptions . . . 400
AIOCommandLineListDevices . . . 401
AIOCommandLineOptionsListDevices . . . 401
AIOCommandLineOverrideADCConfigBlock . . . 402
AIOCommandLineOptionsOverrideADCConfigBlock . . . 402
AIOGetChannelRange . . . 402
AIO_SCRIPTING_OPTIONS . . . 402
AIO_CMDLINE_OPTIONS . . . 402
34.94.2 Variable Documentation . . . 402
opterr . . . 402

optind . . . 402
AIO_DEFAULT_CMDLINE_OPTIONS . . . 402
AIO_DEFAULT_SCRIPTING_OPTIONS . . . 402
34.95 lib/AIOCommandLine.h File Reference . . . 402
34.95.1 Macro Definition Documentation . . . 404
DUMP . . . 404
CNTS . . . 404
JCONF . . . 404
REPEAT . . . 404
34.95.2 Typedef Documentation . . . 404
AIOChannelRangeTmp . . . 404
AIOCommandLineOptions . . . 404
34.95.3 Enumeration Type Documentation . . . 404
DeviceEnum . . . 404
34.95.4 Function Documentation . . . 404
NewDefaultAIOCommandLineOptions . . . 404
NewAIOCommandLineOptionsFromDefaultOptions . . . 404
AIO_CMDLINE_DEFAULT_OPTIONS . . . 404
AIO_CMDLINE_SCRIPTING_OPTIONS . . . 404
AIO_CMDLINE_CLEAR_OPTIONS . . . 404
AIOProcessCmdline . . . 404
AIOProcessCommandLine . . . 405
AIOGetChannelRange . . . 405
AIOPrintUsage . . . 405
DeleteAIOCommandLineOptions . . . 405
AIOCommandLineOptionsListDevices . . . 405
AIOCommandLineOptionsOverrideADCConfigBlock . . . 405
AIO_SCRIPTING_OPTIONS . . . 406
AIO_CMDLINE_OPTIONS . . . 406
AIOCommandLineListDevices . . . 406
AIOCommandLineOverrideADCConfigBlock . . . 406
34.95.5 Variable Documentation . . . 406
AIO_DEFAULT_CMDLINE_OPTIONS . . . 406
AIO_DEFAULT_SCRIPTING_OPTIONS . . . 406
34.96 lib/ndkbuild/jni/AIOCommandLine.h File Reference . . . 406
34.96.1 Macro Definition Documentation . . . 407
DUMP . . . 407
CNTS . . . 407
JCONF . . . 407
REPEAT . . . 407
34.96.2 Typedef Documentation . . . 407
AIOChannelRangeTmp . . . 407
AIOCommandLineOptions . . . 407
34.96.3 Enumeration Type Documentation . . . 408
DeviceEnum . . . 408
34.96.4 Function Documentation . . . 408
NewDefaultAIOCommandLineOptions . . . 408
NewAIOCommandLineOptionsFromDefaultOptions . . . 408
AIO_CMDLINE_DEFAULT_OPTIONS . . . 408

AIO_CMDLINE_SCRIPTING_OPTIONS . . . 408
AIO_CMDLINE_CLEAR_OPTIONS . . . 408
AIOProcessCmdline . . . 408
AIOProcessCommandLine . . . 408
AIOGetChannelRange . . . 409
AIOPrintUsage . . . 409
DeleteAIOCommandLineOptions . . . 409
AIOCommandLineOptionsListDevices . . . 409
AIOCommandLineOptionsOverrideADCConfigBlock . . . 409
AIO_SCRIPTING_OPTIONS . . . 409
AIO_CMDLINE_OPTIONS . . . 409
AIOCommandLineListDevices . . . 409
AIOCommandLineOverrideADCConfigBlock . . . 410
34.96.5 Variable Documentation . . . 410
AIO_DEFAULT_CMDLINE_OPTIONS . . . 410
AIO_DEFAULT_SCRIPTING_OPTIONS . . . 410
34.97 lib/AIOConfiguration.c File Reference . . . 410
34.97.1 Function Documentation . . . 410
NewAIOConfiguration . . . 410
DeleteAIOConfiguration . . . 410
AIOConfigurationInitialize . . . 410
AIOArgumentInitialize . . . 410
AIOConfigurationSetTimeout . . . 410
AIOConfigurationSetDebug . . . 410
34.98 lib/ndkbuild/jni/AIOConfiguration.c File Reference . . . 410
34.98.1 Function Documentation . . . 411
NewAIOConfiguration . . . 411
DeleteAIOConfiguration . . . 411
AIOConfigurationInitialize . . . 411
AIOArgumentInitialize . . . 411
AIOConfigurationSetTimeout . . . 411
AIOConfigurationSetDebug . . . 411
34.99 lib/AIOConfiguration.h File Reference . . . 411
34.99.1 Typedef Documentation . . . 411
AIOConfiguration . . . 411
ADCConfiguration . . . 411
AIOContinousBufConfiguration . . . 411
34.99.2 Enumeration Type Documentation . . . 412
ConfigurationType . . . 412
ADCScanType . . . 412
ADCSetCalFunction . . . 412
34.99.3 Function Documentation . . . 412
AIOConfigurationSetDebug . . . 412
AIOConfigurationSetTimeout . . . 412
NewAIOConfiguration . . . 412
AIOConfigurationInitialize . . . 412
AIOArgumentInitialize . . . 412
34.100 lib/ndkbuild/jni/AIOConfiguration.h File Reference . . . 412
34.100.1 Typedef Documentation . . . 413

AIOConfiguration . . . 413
ADCConfiguration . . . 413
AIOContinousBufConfiguration . . . 413
34.100.2 Enumeration Type Documentation . . . 413
ConfigurationType . . . 413
ADCScanType . . . 413
ADCSetCalFunction . . . 414
34.100.3 Function Documentation . . . 414
AIOConfigurationSetDebug . . . 414
AIOConfigurationSetTimeout . . . 414
NewAIOConfiguration . . . 414
AIOConfigurationInitialize . . . 414
AIOArgumentInitialize . . . 414
34.101 lib/AIOContinuousBuffer.c File Reference . . . 414
34.101.1 Typedef Documentation . . . 417
RangeValueLookup . . . 417
34.101.2 Function Documentation . . . 417
ConvertCountsToVoltsFunction . . . 417
RawCountsWorkFunction . . . 418
AIOContinuousBufForceTerminateAcqusitionOverrun . . . 418
AIOContinuousBufForceTerminateAcqusition . . . 418
NewAIOContinuousBufForCounts . . . 418
NewAIOContinuousBuf . . . 418
AIOContinuousBufGetNumberChannels . . . 418
AIOContinuousBufSetNumberChannels . . . 418
AIOContinuousBufSetBaseSize . . . 418
AIOContinuousBufGetBaseSize . . . 418
AIOContinuousBufGetBufferSize . . . 418
NewAIOContinuousBufForVolts . . . 418
AIOContinuousBuf_InitConfiguration . . . 418
AIOContinuousBufPushN . . . 418
AIOContinuousBufPopN . . . 418
AIOContinuousBufInitADCConfigBlock . . . 418
AIOContinuousBufInitConfiguration . . . 418
AIOContinuousBuf_SendPreConfig . . . 419
AIOContinuousBufSendPreConfig . . . 419
DeleteAIOContinuousBuf . . . 419
AIOContinuousBufSetCountsBuffer . . . 419
AIOContinuousBufSetVoltsBuffer . . . 419
AIOContinuousBufSetStreamingBlockSize . . . 419
AIOContinuousBufGetStreamingBlockSize . . . 419
AIOContinuousBufGetADCConfigBlock . . . 419
AIOContinuousBuf_SetCallback . . . 419
AIOContinuousBufSetCallback . . . 419
AIOContinuousBufSetNumberScans . . . 419
AIOContinuousBufGetNumberScans . . . 419
AIOContinuousBuf_BufSizeForCounts . . . 419
AIOContinuousBufGetUnitSize . . . 419
AIOContinuousBufSetUnitSize . . . 419

AIOContinuousBufReset . . . 419
AIOContinuousBufGetReadPosition . . . 419
AIOContinuousBufGetWritePosition . . . 419
AIOContinuousBufGetRemainingSize . . . 419
AIOContinuousBufGetSize . . . 419
AIOContinuousBufGetSizeNumElements . . . 419
AIOContinuousBufGetStatus . . . 419
AIOContinuousBufPending . . . 419
AIOContinuousBufGetScansRead . . . 419
AIOContinuousBufGetExitCode . . . 419
AIOContinuousBufGetRunStatus . . . 419
AIOContinuousBufCountScansAvailable . . . 419
AIOContinuousBufCountsAvailable . . . 419
AIOContinuousBufGetDataAvailable . . . 419
AIOContinuousBufReadIntegerScanCounts . . . 419
AIOContinuousBufGetNumberOfScansToRead . . . 421
AIOContinuousBufSetNumberOfScansToRead . . . 421
AIOContinuousBufReadIntegerNumberOfScans . . . 421
AIOContinuousBufReadSingle . . . 421
AIOContinuousBufReadCompleteScanCounts . . . 421
AIOContinuousBufGetCallback . . . 421
AIOContinuousBufSetClock . . . 421
AIOContinuousBufGetClock . . . 421
AIOContinuousBufStart . . . 421
AIOContinuousBufStopAcquisition . . . 421
AIOContinuousBufSetChannelMask . . . 421
AIOContinuousBuf_NumberSignals . . . 422
AIOContinuousBufNumberSignals . . . 422
AIOContinuousBuf_NumberChannels . . . 422
AIOContinuousBufNumberChannels . . . 422
AIOContinuousBufWrite . . . 422
AIOContinuousBufWriteCounts . . . 422
AIOContinuousBufGetNumberSamplesPerScan . . . 422
AIOContinuousBufGetTotalSamplesExpected . . . 422
StartStreaming . . . 422
SetConfig . . . 422
ResetCounters . . . 422
AIOContinuousBufLoadCounters . . . 422
AIOContinuousBufCleanup . . . 422
AIOContinuousBufPreSetup . . . 422
AIOContinuousBufNumberSamplesAvailable . . . 422
AIOContinuousBufNumberWriteSamplesRemaining . . . 422
AIOContinuousBufReadNSamples . . . 422
AIOContinuousBufInitiateCallbackAcquisition . . . 422
number_to_read . . . 422
continue_running . . . 422
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction . . . 423
AIOContinuousBufCallbackStart . . . 423
AIOContinuousBufResetDevice . . . 423

AIOContinuousBufRead . . . 423
AIOContinuousBufLock . . . 423
AIOContinuousBufUnlock . . . 424
AIOContinuousBufSimpleSetupConfig . . . 424
AIOContinuousBufEnd . . . 424
AIOContinuousBuf_SetTesting . . . 424
AIOContinuousBufSetTesting . . . 424
AIOContinuousBufGetTesting . . . 424
AIOContinuousBufSetDefaultModeForCounterScan . . . 424
AIOContinuousBufSetDebug . . . 424
AIOContinuousBufGetDebug . . . 424
AIOContinuousBuf_SetDeviceIndex . . . 424
AIOContinuousBufSetDeviceIndex . . . 424
AIOContinuousBuf_SaveConfig . . . 424
AIOContinuousBufSaveConfig . . . 424
AIOContinuousBuf_SetStartAndEndChannel . . . 424
AIOContinuousBufSetStartAndEndChannel . . . 424
AIOContinuousBuf_SetChannelRangeGain . . . 424
AIOContinuousBuf_SetChannelRange . . . 424
AIOContinuousBufSetChannelRange . . . 424
AIOContinuousBufSetTimeout . . . 424
AIOContinuousBufGetTimeout . . . 424
AIOContinuousBuf_SetOversample . . . 424
AIOContinuousBufSetOversample . . . 424
AIOContinuousBufSetOverSample . . . 424
AIOContinuousBuf_GetOverSample . . . 424
AIOContinuousBufGetOversample . . . 424
AIOContinuousBuf_SetAllGainCodeAndDiffMode . . . 424
AIOContinuousBufSetAllGainCodeAndDiffMode . . . 424
AIOContinuousBuf_SetDiscardFirstSample . . . 424
AIOContinuousBufSetDiscardFirstSample . . . 424
AIOContinuousBuf_GetDeviceIndex . . . 424
AIOContinuousBufGetDeviceIndex . . . 424
GetJSONValueOrDefault . . . 425
NewAIOContinuousBufFromJSON . . . 425
AIOContinuousBufToJSON . . . 425
34.101.3 Variable Documentation . . . 425
TrueFalse . . . 425
BaseSizeRange . . . 425
34.102 usb/ndkbuild/jni/AIOContinuousBuffer.c File Reference . . . 425
34.102.1 Typedef Documentation . . . 428
RangeValueLookup . . . 428
34.102.2 Function Documentation . . . 428
ConvertCountsToVoltsFunction . . . 428
RawCountsWorkFunction . . . 428
AIOContinuousBufForceTerminateAcqusitionOverrun . . . 428
AIOContinuousBufForceTerminateAcqusition . . . 428
NewAIOContinuousBufForCounts . . . 428
NewAIOContinuousBuf . . . 428

AIOContinuousBufGetNumberChannels . . . 429
AIOContinuousBufSetNumberChannels . . . 429
AIOContinuousBufSetBaseSize . . . 429
AIOContinuousBufGetBaseSize . . . 429
AIOContinuousBufGetBufferSize . . . 429
NewAIOContinuousBufForVolts . . . 429
AIOContinuousBuf_InitConfiguration . . . 429
AIOContinuousBufPushN . . . 429
AIOContinuousBufPopN . . . 429
AIOContinuousBufInitADCConfigBlock . . . 429
AIOContinuousBufInitConfiguration . . . 429
AIOContinuousBuf_SendPreConfig . . . 429
AIOContinuousBufSendPreConfig . . . 429
DeleteAIOContinuousBuf . . . 429
AIOContinuousBufSetCountsBuffer . . . 429
AIOContinuousBufSetVoltsBuffer . . . 429
AIOContinuousBufSetStreamingBlockSize . . . 429
AIOContinuousBufGetStreamingBlockSize . . . 429
AIOContinuousBufGetADCConfigBlock . . . 429
AIOContinuousBuf_SetCallback . . . 429
AIOContinuousBufSetCallback . . . 429
AIOContinuousBufSetNumberScans . . . 429
AIOContinuousBufGetNumberScans . . . 429
AIOContinuousBuf_BufSizeForCounts . . . 429
AIOContinuousBufGetUnitSize . . . 429
AIOContinuousBufSetUnitSize . . . 429
AIOContinuousBufReset . . . 430
AIOContinuousBufGetReadPosition . . . 430
AIOContinuousBufGetWritePosition . . . 430
AIOContinuousBufGetRemainingSize . . . 430
AIOContinuousBufGetSize . . . 430
AIOContinuousBufGetSizeNumElements . . . 430
AIOContinuousBufGetStatus . . . 430
AIOContinuousBufPending . . . 430
AIOContinuousBufGetScansRead . . . 430
AIOContinuousBufGetExitCode . . . 430
AIOContinuousBufGetRunStatus . . . 430
AIOContinuousBufCountScansAvailable . . . 430
AIOContinuousBufCountsAvailable . . . 430
AIOContinuousBufGetDataAvailable . . . 430
AIOContinuousBufReadIntegerScanCounts . . . 430
AIOContinuousBufGetNumberOfScansToRead . . . 430
AIOContinuousBufSetNumberOfScansToRead . . . 430
AIOContinuousBufReadIntegerNumberOfScans . . . 430
AIOContinuousBufReadSingle . . . 431
AIOContinuousBufReadCompleteScanCounts . . . 431
AIOContinuousBufGetCallback . . . 431
AIOContinuousBufSetClock . . . 431
AIOContinuousBufGetClock . . . 431

AIOContinuousBufStart . . . 431
AIOContinuousBufStopAcquisition . . . 431
AIOContinuousBufSetChannelMask . . . 431
AIOContinuousBuf_NumberSignals . . . 431
AIOContinuousBufNumberSignals . . . 431
AIOContinuousBuf_NumberChannels . . . 431
AIOContinuousBufNumberChannels . . . 431
AIOContinuousBufWrite . . . 431
AIOContinuousBufWriteCounts . . . 432
AIOContinuousBufGetNumberSamplesPerScan . . . 432
AIOContinuousBufGetTotalSamplesExpected . . . 432
StartStreaming . . . 432
SetConfig . . . 432
ResetCounters . . . 432
AIOContinuousBufLoadCounters . . . 432
AIOContinuousBufCleanup . . . 432
AIOContinuousBufPreSetup . . . 432
AIOContinuousBufNumberSamplesAvailable . . . 432
AIOContinuousBufNumberWriteSamplesRemaining . . . 432
AIOContinuousBufReadNSamples . . . 432
AIOContinuousBufInitiateCallbackAcquisition . . . 432
number_to_read . . . 432
continue_running . . . 432
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction . . . 432
AIOContinuousBufCallbackStart . . . 432
AIOContinuousBufResetDevice . . . 433
AIOContinuousBufRead . . . 433
AIOContinuousBufLock . . . 433
AIOContinuousBufUnlock . . . 433
AIOContinuousBufSimpleSetupConfig . . . 433
AIOContinuousBufEnd . . . 433
AIOContinuousBuf_SetTesting . . . 433
AIOContinuousBufSetTesting . . . 433
AIOContinuousBufGetTesting . . . 433
AIOContinuousBufSetDefaultModeForCounterScan . . . 433
AIOContinuousBufSetDebug . . . 433
AIOContinuousBufGetDebug . . . 433
AIOContinuousBuf_SetDeviceIndex . . . 433
AIOContinuousBufSetDeviceIndex . . . 433
AIOContinuousBuf_SaveConfig . . . 433
AIOContinuousBufSaveConfig . . . 433
AIOContinuousBuf_SetStartAndEndChannel . . . 433
AIOContinuousBufSetStartAndEndChannel . . . 433
AIOContinuousBuf_SetChannelRangeGain . . . 433
AIOContinuousBuf_SetChannelRange . . . 434
AIOContinuousBufSetChannelRange . . . 434
AIOContinuousBufSetTimeout . . . 434
AIOContinuousBufGetTimeout . . . 434
AIOContinuousBuf_SetOversample . . . 434

AIOContinuousBufSetOversample . . . 434
AIOContinuousBufSetOverSample . . . 434
AIOContinuousBuf_GetOverSample . . . 434
AIOContinuousBufGetOversample . . . 434
AIOContinuousBuf_SetAllGainCodeAndDiffMode . . . 434
AIOContinuousBufSetAllGainCodeAndDiffMode . . . 434
AIOContinuousBuf_SetDiscardFirstSample . . . 434
AIOContinuousBufSetDiscardFirstSample . . . 434
AIOContinuousBuf_GetDeviceIndex . . . 434
AIOContinuousBufGetDeviceIndex . . . 434
GetJSONValueOrDefault . . . 434
NewAIOContinuousBufFromJSON . . . 434
AIOContinuousBufToJSON . . . 434
34.102.3 Variable Documentation . . . 434
TrueFalse . . . 434
BaseSizeRange . . . 434
34.103 lib/AIOContinuousBuffer.h File Reference . . . 435
34.103.1 Macro Definition Documentation . . . 437
ROOTCLOCK . . . 437
34.103.2 Typedef Documentation . . . 437
AIOUSB_WorkFn . . . 437
AIOContinuousBuf . . . 437
34.103.3 Enumeration Type Documentation . . . 437
AIO_CONT_BUF_TYPE . . . 437
34.103.4 Function Documentation . . . 437
NewAIOContinuousBuf . . . 437
NewAIOContinuousBufForCounts . . . 438
NewAIOContinuousBufForVolts . . . 438
DeleteAIOContinuousBuf . . . 438
AIOContinuousBufInitConfiguration . . . 438
AIOContinuousBufInitADCConfigBlock . . . 438
AIOContinuousBufGetCallback . . . 438
AIOContinuousBufSetCallback . . . 438
AIOContinuousBufSetStreamingBlockSize . . . 438
AIOContinuousBufGetStreamingBlockSize . . . 438
AIOContinuousBufGetADCConfigBlock . . . 438
AIOContinuousBufSetNumberChannels . . . 438
AIOContinuousBufGetNumberChannels . . . 438
AIOContinuousBufGetOversample . . . 438
AIOContinuousBufSetOversample . . . 438
AIOContinuousBufNumberChannels . . . 438
AIOContinuousBufSetBaseSize . . . 438
AIOContinuousBufGetBaseSize . . . 438
AIOContinuousBufGetBufferSize . . . 439
AIOContinuousBufSetUnitSize . . . 439
AIOContinuousBufGetUnitSize . . . 439
AIOContinuousBufSetTesting . . . 439
AIOContinuousBufGetTesting . . . 439
AIOContinuousBufSendPreConfig . . . 439

AIOContinuousBufSetStartAndEndChannel . . . 439
AIOContinuousBufSetAllGainCodeAndDiffMode . . . 439
AIOContinuousBufGetDeviceIndex . . . 439
AIOContinuousBufSetDiscardFirstSample . . . 439
AIOContinuousBufSetChannelMask . . . 439
AIOContinuousBufNumberSignals . . . 439
AIOContinuousBufSetChannelRange . . . 439
AIOContinuousBufSaveConfig . . . 439
AIOContinuousBufSetDeviceIndex . . . 439
AIOContinuousBufResetDevice . . . 439
AIOContinuousBufSetTimeout . . . 439
AIOContinuousBufGetTimeout . . . 439
AIOContinuousBufSetDebug . . . 439
AIOContinuousBufGetDebug . . . 439
AIOContinuousBufGetNumberScans . . . 439
AIOContinuousBufSetNumberScans . . . 439
AIOContinuousBufNumberWriteSamplesRemaining . . . 439
AIOContinuousBufNumberSamplesAvailable . . . 439
AIOContinuousBufGetNumberSamplesPerScan . . . 439
AIOContinuousBufGetTotalSamplesExpected . . . 439
AIOContinuousBufReset . . . 439
AIOContinuousBufPushN . . . 440
AIOContinuousBufPopN . . . 440
AIOContinuousBufLock . . . 440
AIOContinuousBufUnlock . . . 440
AIOContinuousBufCallbackStart . . . 440
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction . . . 440
AIOContinuousBufStopAcquisition . . . 441
AIOContinuousBufInitiateCallbackAcquisition . . . 441
AIOContinuousBufGetReadPosition . . . 441
AIOContinuousBufGetWritePosition . . . 441
AIOContinuousBufGetRemainingSize . . . 441
AIOContinuousBufGetStatus . . . 441
AIOContinuousBufGetExitCode . . . 441
AIOContinuousBufGetRunStatus . . . 441
AIOContinuousBufPending . . . 441
AIOContinuousBufGetScansRead . . . 441
AIOContinuousBufReadIntegerScanCounts . . . 441
AIOContinuousBufReadCompleteScanCounts . . . 441
AIOContinuousBufReadIntegerNumberOfScans . . . 441
AIOContinuousBufSetCountsBuffer . . . 441
AIOContinuousBufSetVoltsBuffer . . . 441
AIOContinuousBufCountScansAvailable . . . 441
AIOContinuousBufSetClock . . . 441
AIOContinuousBufGetClock . . . 441
AIOContinuousBufEnd . . . 441
AIOContinuousBufSimpleSetupConfig . . . 442
AIOContinuousBufRead . . . 442
AIOContinuousBufWrite . . . 442

AIOContinuousBufWriteCounts . . . 442
AIOContinuousBufCleanup . . . 442
AIOContinuousBufToJSON . . . 442
NewAIOContinuousBufFromJSON . . . 442
34.104 lib/ndkbuild/jni/AIOContinuousBuffer.h File Reference . . . 442
34.104.1 Macro Definition Documentation . . . 445
ROOTCLOCK . . . 445
34.104.2 Typedef Documentation . . . 445
AIOUSB_WorkFn . . . 445
AIOContinuousBuf . . . 445
34.104.3 Enumeration Type Documentation . . . 445
AIO_CONT_BUF_TYPE . . . 445
34.104.4 Function Documentation . . . 445
NewAIOContinuousBuf . . . 445
NewAIOContinuousBufForCounts . . . 445
NewAIOContinuousBufForVolts . . . 445
DeleteAIOContinuousBuf . . . 445
AIOContinuousBufInitConfiguration . . . 445
AIOContinuousBufInitADCConfigBlock . . . 446
AIOContinuousBufGetCallback . . . 446
AIOContinuousBufSetCallback . . . 446
AIOContinuousBufSetStreamingBlockSize . . . 446
AIOContinuousBufGetStreamingBlockSize . . . 446
AIOContinuousBufGetADCConfigBlock . . . 446
AIOContinuousBufSetNumberChannels . . . 446
AIOContinuousBufGetNumberChannels . . . 446
AIOContinuousBufGetOversample . . . 446
AIOContinuousBufSetOversample . . . 446
AIOContinuousBufNumberChannels . . . 446
AIOContinuousBufSetBaseSize . . . 446
AIOContinuousBufGetBaseSize . . . 446
AIOContinuousBufGetBufferSize . . . 446
AIOContinuousBufSetUnitSize . . . 446
AIOContinuousBufGetUnitSize . . . 446
AIOContinuousBufSetTesting . . . 446
AIOContinuousBufGetTesting . . . 446
AIOContinuousBufSendPreConfig . . . 446
AIOContinuousBufSetStartAndEndChannel . . . 446
AIOContinuousBufSetAllGainCodeAndDiffMode . . . 446
AIOContinuousBufGetDeviceIndex . . . 446
AIOContinuousBufSetDiscardFirstSample . . . 447
AIOContinuousBufSetChannelMask . . . 447
AIOContinuousBufNumberSignals . . . 448
AIOContinuousBufSetChannelRange . . . 448
AIOContinuousBufSaveConfig . . . 448
AIOContinuousBufSetDeviceIndex . . . 448
AIOContinuousBufResetDevice . . . 448
AIOContinuousBufSetTimeout . . . 448
AIOContinuousBufGetTimeout . . . 448

AIOContinuousBufSetDebug . . . 448
AIOContinuousBufGetDebug . . . 448
AIOContinuousBufGetNumberScans . . . 448
AIOContinuousBufSetNumberScans . . . 448
AIOContinuousBufNumberWriteSamplesRemaining . . . 448
AIOContinuousBufNumberSamplesAvailable . . . 448
AIOContinuousBufGetNumberSamplesPerScan . . . 448
AIOContinuousBufGetTotalSamplesExpected . . . 448
AIOContinuousBufReset . . . 448
AIOContinuousBufPushN . . . 448
AIOContinuousBufPopN . . . 448
AIOContinuousBufLock . . . 448
AIOContinuousBufUnlock . . . 448
AIOContinuousBufCallbackStart . . . 448
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction . . . 449
AIOContinuousBufStopAcquisition . . . 449
AIOContinuousBufInitiateCallbackAcquisition . . . 449
AIOContinuousBufGetReadPosition . . . 449
AIOContinuousBufGetWritePosition . . . 449
AIOContinuousBufGetRemainingSize . . . 449
AIOContinuousBufGetStatus . . . 449
AIOContinuousBufGetExitCode . . . 449
AIOContinuousBufGetRunStatus . . . 449
AIOContinuousBufPending . . . 449
AIOContinuousBufGetScansRead . . . 449
AIOContinuousBufReadIntegerScanCounts . . . 449
AIOContinuousBufReadCompleteScanCounts . . . 450
AIOContinuousBufReadIntegerNumberOfScans . . . 450
AIOContinuousBufSetCountsBuffer . . . 450
AIOContinuousBufSetVoltsBuffer . . . 450
AIOContinuousBufCountScansAvailable . . . 450
AIOContinuousBufSetClock . . . 450
AIOContinuousBufGetClock . . . 450
AIOContinuousBufEnd . . . 450
AIOContinuousBufSimpleSetupConfig . . . 450
AIOContinuousBufRead . . . 450
AIOContinuousBufWrite . . . 450
AIOContinuousBufWriteCounts . . . 451
AIOContinuousBufCleanup . . . 451
AIOContinuousBufToJSON . . . 451
NewAIOContinuousBufFromJSON . . . 451
34.105 usb/AIOCountsConverter.c File Reference . . . 451
34.105.1 Function Documentation . . . 451
default_out . . . 451
enhanced_out . . . 451
NewAIOCountsConverterWithBuffer . . . 451
NewAIOCountsConverterWithScanLimiter . . . 451
NewAIOCountsConverter . . . 452
DeleteAIOCountsConverter . . . 452

AIOCountsConverterReset . . . 452
AIOCountsConverterConvertNScans . . . 452
AIOCountsConverterConvertAllAvailableScans . . . 452
Convert . . . 452
AIOCountsConverterConvertFifo . . . 452
AIOCountsConverterConvert . . . 452
NewAIOGainRangeFromADCConfigBlock . . . 452
DeleteAIOGainRange . . . 452
34.106 /lib/ndkbuild/jni/AIOCountsConverter.c File Reference . . . 452
34.106.1 Function Documentation . . . 453
default_out . . . 453
enhanced_out . . . 453
NewAIOCountsConverterWithBuffer . . . 453
NewAIOCountsConverterWithScanLimiter . . . 453
NewAIOCountsConverter . . . 453
DeleteAIOCountsConverter . . . 453
AIOCountsConverterReset . . . 453
AIOCountsConverterConvertNScans . . . 453
AIOCountsConverterConvertAllAvailableScans . . . 453
Convert . . . 453
AIOCountsConverterConvertFifo . . . 453
AIOCountsConverterConvert . . . 453
NewAIOGainRangeFromADCConfigBlock . . . 453
DeleteAIOGainRange . . . 453
34.107 /lib/AIOCountsConverter.h File Reference . . . 453
34.107.1 Typedef Documentation . . . 454
AIOCountsConverter . . . 454
34.107.2 Function Documentation . . . 454
NewAIOCountsConverterWithBuffer . . . 454
NewAIOCountsConverter . . . 454
NewAIOCountsConverterFromAIOContinuousBuf . . . 454
NewAIOCountsConverterWithScanLimiter . . . 454
AIOCountsConverterReset . . . 454
DeleteAIOCountsConverter . . . 454
AIOCountsConverterConvertNScans . . . 454
AIOCountsConverterConvertAllAvailableScans . . . 454
AIOCountsConverterConvert . . . 454
AIOCountsConverterConvertFifo . . . 454
NewAIOGainRangeFromADCConfigBlock . . . 454
DeleteAIOGainRange . . . 454
34.108 /lib/ndkbuild/jni/AIOCountsConverter.h File Reference . . . 454
34.108.1 Typedef Documentation . . . 455
AIOCountsConverter . . . 455
34.108.2 Function Documentation . . . 455
NewAIOCountsConverterWithBuffer . . . 455
NewAIOCountsConverter . . . 455
NewAIOCountsConverterFromAIOContinuousBuf . . . 455
NewAIOCountsConverterWithScanLimiter . . . 455
AIOCountsConverterReset . . . 455

DeleteAIOCountsConverter . . . 455
AIOCountsConverterConvertNScans . . . 455
AIOCountsConverterConvertAllAvailableScans . . . 455
AIOCountsConverterConvert . . . 455
AIOCountsConverterConvertFifo . . . 455
NewAIOGainRangeFromADCConfigBlock . . . 456
DeleteAIOGainRange . . . 456
34.109 lib/AIODeviceInfo.c File Reference . . . 456
34.109.1 Function Documentation . . . 456
NewAIODeviceInfo . . . 456
DeleteAIODeviceInfo . . . 456
AIODeviceInfoGetName . . . 456
AIODeviceInfoGetCounters . . . 456
AIODeviceInfoGetDIOBytes . . . 456
AIODeviceInfoGet . . . 456
34.110 lib/ndkbuild/jni/AIODeviceInfo.c File Reference . . . 456
34.110.1 Function Documentation . . . 457
NewAIODeviceInfo . . . 457
DeleteAIODeviceInfo . . . 457
AIODeviceInfoGetName . . . 457
AIODeviceInfoGetCounters . . . 457
AIODeviceInfoGetDIOBytes . . . 457
AIODeviceInfoGet . . . 457
34.111 lib/AIODeviceInfo.h File Reference . . . 457
34.111.1 Typedef Documentation . . . 457
AIODeviceInfo . . . 457
34.111.2 Function Documentation . . . 457
NewAIODeviceInfo . . . 457
DeleteAIODeviceInfo . . . 457
AIODeviceInfoGetName . . . 457
AIODeviceInfoGet . . . 457
AIODeviceInfoGetCounters . . . 457
AIODeviceInfoGetDIOBytes . . . 457
34.112 lib/ndkbuild/jni/AIODeviceInfo.h File Reference . . . 457
34.112.1 Typedef Documentation . . . 458
AIODeviceInfo . . . 458
34.112.2 Function Documentation . . . 458
NewAIODeviceInfo . . . 458
DeleteAIODeviceInfo . . . 458
AIODeviceInfoGetName . . . 458
AIODeviceInfoGet . . . 458
AIODeviceInfoGetCounters . . . 458
AIODeviceInfoGetDIOBytes . . . 458
34.113 lib/AIODeviceQuery.c File Reference . . . 458
34.113.1 Function Documentation . . . 459
NewAIODeviceQuery . . . 459
DeleteAIODeviceQuery . . . 460
AIODeviceQueryToStr . . . 460
AIODeviceQueryToRepr . . . 460

AIODeviceQueryGetProductID . . . 460
AIODeviceQueryGetIndex . . . 460
AIODeviceQueryNameSize . . . 460
AIODeviceQueryGetName . . . 461
AIODeviceQueryGetNumDIOBytes . . . 461
AIODeviceQueryGetNumCounters . . . 461
34.114 lib/ndkbuild/jni/AIODeviceQuery.c File Reference . . . 461
34.114.1 Function Documentation . . . 462
NewAIODeviceQuery . . . 462
DeleteAIODeviceQuery . . . 462
AIODeviceQueryToStr . . . 462
AIODeviceQueryToRepr . . . 462
AIODeviceQueryGetProductID . . . 462
AIODeviceQueryGetIndex . . . 463
AIODeviceQueryNameSize . . . 464
AIODeviceQueryGetName . . . 464
AIODeviceQueryGetNumDIOBytes . . . 464
AIODeviceQueryGetNumCounters . . . 464
34.115 lib/AIODeviceQuery.h File Reference . . . 464
34.115.1 Typedef Documentation . . . 465
AIODeviceQuery . . . 465
34.115.2 Function Documentation . . . 465
NewAIODeviceQuery . . . 465
DeleteAIODeviceQuery . . . 465
AIODeviceQueryToStr . . . 465
AIODeviceQueryToRepr . . . 466
AIODeviceQueryGetProductID . . . 466
AIODeviceQueryNameSize . . . 466
AIODeviceQueryGetName . . . 466
AIODeviceQueryGetNumDIOBytes . . . 466
AIODeviceQueryGetNumCounters . . . 466
AIODeviceQueryGetIndex . . . 467
34.116 lib/ndkbuild/jni/AIODeviceQuery.h File Reference . . . 467
34.116.1 Typedef Documentation . . . 468
AIODeviceQuery . . . 468
34.116.2 Function Documentation . . . 468
NewAIODeviceQuery . . . 468
DeleteAIODeviceQuery . . . 468
AIODeviceQueryToStr . . . 468
AIODeviceQueryToRepr . . . 468
AIODeviceQueryGetProductID . . . 468
AIODeviceQueryNameSize . . . 468
AIODeviceQueryGetName . . . 469
AIODeviceQueryGetNumDIOBytes . . . 469
AIODeviceQueryGetNumCounters . . . 469
AIODeviceQueryGetIndex . . . 469
34.117 lib/AIODeviceTable.c File Reference . . . 469
34.117.1 Macro Definition Documentation . . . 470
NUM_PROD_NAMES . . . 470

34.117.2 Function Documentation . . . 470
AIOUSB_SetInit . . . 470
AIODeviceTableInit . . . 470
AIOUSB_IsInit . . . 470
AIOUSB_InitTest . . . 470
AIOUSB_Cleanup . . . 470
QueryDeviceInfo . . . 470
ProductIDToName . . . 471
ProductNameToID . . . 471
GetDevices . . . 471
AIODeviceTableGetUSBDeviceAtIndex . . . 471
GetSafeDeviceName . . . 472
AIOUSB_EnsureOpen . . . 472
AIODeviceTableGetDeviceAtIndex . . . 472
AIODeviceTableGetAIOUSBDeviceAtIndex . . . 472
AIOUSBGetError . . . 472
AIODeviceTableAddDeviceToDeviceTable . . . 472
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice . . . 472
ClearAIODeviceTable . . . 472
AIODeviceTableSetDeviceID . . . 472
AIOUSB_GetAllDevices . . . 472
AIODeviceTablePopulateTableTest . . . 472
CloseAllDevices . . . 472
AIODeviceTableClearDevices . . . 472
ClearDevices . . . 472
AIODeviceTablePopulateTable . . . 472
AIOUSB_Init . . . 473
AIOUSB_Exit . . . 473
AIOUSB_Reset . . . 473
34.117.3 Variable Documentation . . . 473
deviceTable . . . 473
AIOUSB_INIT_PATTERN . . . 473
aiousbInit . . . 473
34.118 lib/ndkbuild/jni/AIODeviceTable.c File Reference . . . 473
34.118.1 Macro Definition Documentation . . . 474
NUM_PROD_NAMES . . . 474
34.118.2 Function Documentation . . . 474
AIOUSB_SetInit . . . 474
AIODeviceTableInit . . . 474
AIOUSB_IsInit . . . 474
AIOUSB_InitTest . . . 474
AIOUSB_Cleanup . . . 474
QueryDeviceInfo . . . 474
ProductIDToName . . . 474
ProductNameToID . . . 475
GetDevices . . . 475
AIODeviceTableGetUSBDeviceAtIndex . . . 475
GetSafeDeviceName . . . 475
AIOUSB_EnsureOpen . . . 475

AIODeviceTableGetDeviceAtIndex . . . 476
AIODeviceTableGetAIOUSBDeviceAtIndex . . . 476
AIOUSBGetError . . . 476
AIODeviceTableAddDeviceToDeviceTable . . . 476
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice . . . 476
ClearAIODeviceTable . . . 476
AIODeviceTableSetDeviceID . . . 476
AIOUSB_GetAllDevices . . . 476
AIODeviceTablePopulateTableTest . . . 476
CloseAllDevices . . . 476
AIODeviceTableClearDevices . . . 476
ClearDevices . . . 476
AIODeviceTablePopulateTable . . . 476
AIOUSB_Init . . . 476
AIOUSB_Exit . . . 476
AIOUSB_Reset . . . 476
34.118.3 Variable Documentation . . . 476
deviceTable . . . 476
AIOUSB_INIT_PATTERN . . . 476
aiousbInit . . . 476
34.119 lib/AIODeviceTable.h File Reference . . . 477
34.119.1 Function Documentation . . . 477
AIODeviceTableAddDeviceToDeviceTable . . . 477
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice . . . 478
AIODeviceTablePopulateTable . . . 478
AIODeviceTablePopulateTableTest . . . 478
AIODeviceTableClearDevices . . . 478
ClearDevices . . . 478
AIODeviceTableGetDeviceAtIndex . . . 478
AIODeviceTableGetAIOUSBDeviceAtIndex . . . 478
AIODeviceTableGetUSBDeviceAtIndex . . . 478
QueryDeviceInfo . . . 478
GetDevices . . . 478
GetSafeDeviceName . . . 479
ProductIDToName . . . 479
ProductNameToID . . . 479
AIOUSB_Init . . . 479
AIOUSB_EnsureOpen . . . 479
AIOUSB_IsInit . . . 480
AIOUSB_Exit . . . 480
AIOUSB_Reset . . . 480
AIODeviceTableInit . . . 480
ClearAIODeviceTable . . . 480
CloseAllDevices . . . 480
AIOUSB_GetAllDevices . . . 480
AIOUSBGetError . . . 480
34.119.2 Variable Documentation . . . 480
deviceTable . . . 480
AIOUSB_INIT_PATTERN . . . 480

34.120 lib/ndkbuild/jni/AIODeviceTable.h File Reference . . . 480
34.120.1 Function Documentation . . . 481
AIODeviceTableAddDeviceToDeviceTable . . . 481
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice . . . 481
AIODeviceTablePopulateTable . . . 481
AIODeviceTablePopulateTableTest . . . 481
AIODeviceTableClearDevices . . . 481
ClearDevices . . . 481
AIODeviceTableGetDeviceAtIndex . . . 481
AIODeviceTableGetAIOUSBDeviceAtIndex . . . 481
AIODeviceTableGetUSBDeviceAtIndex . . . 481
QueryDeviceInfo . . . 482
GetDevices . . . 482
GetSafeDeviceName . . . 482
ProductIDToName . . . 482
ProductNameToID . . . 483
AIOUSB_Init . . . 483
AIOUSB_EnsureOpen . . . 483
AIOUSB_IsInit . . . 483
AIOUSB_Exit . . . 483
AIOUSB_Reset . . . 483
AIODeviceTableInit . . . 483
ClearAIODeviceTable . . . 483
CloseAllDevices . . . 484
AIOUSB_GetAllDevices . . . 484
AIOUSBGetError . . . 484
34.120.2 Variable Documentation . . . 484
deviceTable . . . 484
AIOUSB_INIT_PATTERN . . . 484
34.121 lib/AIOEither.c File Reference . . . 484
34.121.1 Macro Definition Documentation . . . 484
LOOKUP . . . 484
AIO_EITHER_CHECK_VALUE . . . 484
AIO_EITHER_GET_VALUE . . . 484
34.121.2 Function Documentation . . . 485
AIOEitherClear . . . 485
AIOEitherSetRight . . . 485
AIOEitherGetRight . . . 485
AIOEitherSetLeft . . . 485
AIOEitherGetLeft . . . 485
AIOEitherHasError . . . 485
AIOEitherToString . . . 485
AIOEitherToInt . . . 485
AIOEitherToShort . . . 485
AIOEitherToUnsigned . . . 485
AIOEitherToDouble . . . 485
AIOEitherToAIONumber . . . 485
AIOEitherToAIORetType . . . 485
34.122 lib/ndkbuild/jni/AIOEither.c File Reference . . . 485

34.122.1 Macro Definition Documentation . . . 485
LOOKUP . . . 485
AIO_EITHER_CHECK_VALUE . . . 485
AIO_EITHER_GET_VALUE . . . 486
34.122.2 Function Documentation . . . 486
AIOEitherClear . . . 486
AIOEitherSetRight . . . 486
AIOEitherGetRight . . . 486
AIOEitherSetLeft . . . 486
AIOEitherGetLeft . . . 486
AIOEitherHasError . . . 486
AIOEitherToString . . . 486
AIOEitherToInt . . . 486
AIOEitherToShort . . . 486
AIOEitherToUnsigned . . . 486
AIOEitherToDouble . . . 486
AIOEitherToAIONumber . . . 486
AIOEitherToAIORetType . . . 486
34.123 lib/AIOEither.h File Reference . . . 486
34.123.1 Macro Definition Documentation . . . 487
AIO_ERROR_VALUE . . . 487
34.123.2 Typedef Documentation . . . 487
AIO_EITHER_VALUE_ITEM . . . 487
AIOEither . . . 487
34.123.3 Enumeration Type Documentation . . . 487
AIO_EITHER_TYPE . . . 487
34.123.4 Function Documentation . . . 488
AIOEitherClear . . . 488
AIOEitherSetRight . . . 488
AIOEitherGetRight . . . 488
AIOEitherSetLeft . . . 488
AIOEitherGetLeft . . . 488
AIOEitherHasError . . . 488
AIOEitherToString . . . 488
AIOEitherToInt . . . 488
AIOEitherToShort . . . 488
AIOEitherToUnsigned . . . 488
AIOEitherToDouble . . . 488
AIOEitherToAIONumber . . . 488
AIOEitherToAIORetType . . . 488
34.124 lib/ndkbuild/jni/AIOEither.h File Reference . . . 488
34.124.1 Macro Definition Documentation . . . 489
AIO_ERROR_VALUE . . . 489
34.124.2 Typedef Documentation . . . 489
AIO_EITHER_VALUE_ITEM . . . 489
AIOEither . . . 489
34.124.3 Enumeration Type Documentation . . . 489
AIO_EITHER_TYPE . . . 489
34.124.4 Function Documentation . . . 490

AIOEitherClear . . . 490
AIOEitherSetRight . . . 490
AIOEitherGetRight . . . 490
AIOEitherSetLeft . . . 490
AIOEitherGetLeft . . . 490
AIOEitherHasError . . . 490
AIOEitherToString . . . 490
AIOEitherToInt . . . 490
AIOEitherToShort . . . 490
AIOEitherToUnsigned . . . 490
AIOEitherToDouble . . . 490
AIOEitherToAIONumber . . . 490
AIOEitherToAIORetType . . . 490
34.125 usb/AIOFifo.c File Reference . . . 490
34.125.1 Macro Definition Documentation . . . 491
LOOKUP . . . 491
34.125.2 Function Documentation . . . 491
delta . . . 491
AIOFifoWriteSizeRemaining . . . 491
AIOFifoWriteSizeRemainingNumElements . . . 491
AIOFifoGetSize . . . 491
AIOFifoGetSizeNumElements . . . 491
rdelta . . . 491
AIOFifoReadSize . . . 491
AIOFifoReadSizeNumElements . . . 491
AIOFifoResize . . . 491
AIOFifoInitialize . . . 491
NewAIOFifo . . . 491
AIOFifoAllOrNoneInitialize . . . 491
NewAIOFifoAllOrNone . . . 491
AIOFifoReset . . . 491
AIOFifoGetRefSize . . . 491
Push . . . 491
PushN . . . 491
Pop . . . 491
PopN . . . 491
NewAIOFifoTYPE . . . 491
DeleteAIOFifoTYPE . . . 491
DeleteAIOFifo . . . 491
increment . . . 491
AIOFifoRead . . . 491
AIOFifoWrite . . . 491
AIOFifoWriteAllOrNone . . . 492
AIOFifoReadAllOrNone . . . 492
AIOFifoReadPosition . . . 492
AIOFifoWritePosition . . . 492
TEMPLATE_AIOFIFO_API . . . 492
TEMPLATE_AIOFIFO_API . . . 492
34.126 usb/ndkbuild/jni/AIOFifo.c File Reference . . . 492

34.126.1 Macro Definition Documentation . . . 493
LOOKUP . . . 493
34.126.2 Function Documentation . . . 493
delta . . . 493
AIOFifoWriteSizeRemaining . . . 493
AIOFifoWriteSizeRemainingNumElements . . . 493
AIOFifoGetSize . . . 493
AIOFifoGetSizeNumElements . . . 493
rdelta . . . 493
AIOFifoReadSize . . . 493
AIOFifoReadSizeNumElements . . . 493
AIOFifoResize . . . 493
AIOFifoInitialize . . . 493
NewAIOFifo . . . 493
AIOFifoAllOrNoneInitialize . . . 493
NewAIOFifoAllOrNone . . . 493
AIOFifoReset . . . 493
AIOFifoGetRefSize . . . 493
Push . . . 493
PushN . . . 493
Pop . . . 493
PopN . . . 493
NewAIOFifoTYPE . . . 493
DeleteAIOFifoTYPE . . . 493
DeleteAIOFifo . . . 493
increment . . . 493
AIOFifoRead . . . 493
AIOFifoWrite . . . 493
AIOFifoWriteAllOrNone . . . 493
AIOFifoReadAllOrNone . . . 493
AIOFifoReadPosition . . . 493
AIOFifoWritePosition . . . 493
TEMPLATE_AIOFIFO_API . . . 493
TEMPLATE_AIOFIFO_API . . . 494
34.127 Tb/AIOFifo.h File Reference . . . 494
34.127.1 Macro Definition Documentation . . . 495
LOCKING_MECHANISM . . . 495
GRAB_RESOURCE . . . 495
RELEASE_RESOURCE . . . 495
AIO_FIFO_INTERFACE . . . 495
TEMPLATE_AIOFIFO_INTERFACE . . . 495
TEMPLATE_AIOFIFO_API . . . 495
34.127.2 Typedef Documentation . . . 495
AIOFifo . . . 495
TYPE . . . 495
INPUT_TYPE . . . 495
AIOFifoTYPE . . . 495
34.127.3 Function Documentation . . . 495
TEMPLATE_AIOFIFO_INTERFACE . . . 495

TEMPLATE_AIOFIFO_INTERFACE . . . 496
NewAIOFifo . . . 496
DeleteAIOFifo . . . 496
AIOFifoReset . . . 496
AIOFifoRead . . . 496
AIOFifoWrite . . . 496
AIOFifoWriteAllOrNone . . . 496
AIOFifoReadAllOrNone . . . 496
AIOFifoGetRefSize . . . 496
NewAIOFifoTYPE . . . 496
Push . . . 496
PushN . . . 496
PopN . . . 496
AIOFifoWriteSizeRemaining . . . 496
AIOFifoWriteSizeRemainingNumElements . . . 496
AIOFifoReadSize . . . 496
AIOFifoReadSizeNumElements . . . 496
AIOFifoGetSize . . . 496
AIOFifoGetSizeNumElements . . . 496
AIOFifoResize . . . 496
AIOFifoReadPosition . . . 496
AIOFifoWritePosition . . . 496
34.128 lib/ndkbuild/jni/AIOFifo.h File Reference . . . 496
34.128.1 Macro Definition Documentation . . . 497
LOCKING_MECHANISM . . . 497
GRAB_RESOURCE . . . 497
RELEASE_RESOURCE . . . 497
AIO_FIFO_INTERFACE . . . 497
TEMPLATE_AIOFIFO_INTERFACE . . . 498
TEMPLATE_AIOFIFO_API . . . 498
34.128.2 Typedef Documentation . . . 498
AIOFifo . . . 498
TYPE . . . 498
INPUT_TYPE . . . 498
AIOFifoTYPE . . . 498
34.128.3 Function Documentation . . . 498
TEMPLATE_AIOFIFO_INTERFACE . . . 498
TEMPLATE_AIOFIFO_INTERFACE . . . 498
NewAIOFifo . . . 498
DeleteAIOFifo . . . 498
AIOFifoReset . . . 498
AIOFifoRead . . . 498
AIOFifoWrite . . . 498
AIOFifoWriteAllOrNone . . . 498
AIOFifoReadAllOrNone . . . 498
AIOFifoGetRefSize . . . 498
NewAIOFifoTYPE . . . 498
Push . . . 499
PushN . . . 499

PopN . . . 499
AIOFifoWriteSizeRemaining . . . 499
AIOFifoWriteSizeRemainingNumElements . . . 499
AIOFifoReadSize . . . 499
AIOFifoReadSizeNumElements . . . 499
AIOFifoGetSize . . . 499
AIOFifoGetSizeNumElements . . . 499
AIOFifoResize . . . 499
AIOFifoReadPosition . . . 499
AIOFifoWritePosition . . . 499
34.129 lib/AIOList.c File Reference . . . 499
34.129.1 Function Documentation . . . 499
intToString . . . 499
Deleteint . . . 499
TAIL_Q_LIST_IMPLEMENTATION . . . 499
TAIL_Q_LIST_IMPLEMENTATION . . . 499
intlistFirst . . . 499
Newintlist . . . 500
Deleteintlist . . . 500
intlistToString . . . 500
intlistSize . . . 500
intlistInsert . . . 500
34.129.2 Variable Documentation . . . 500
newone . . . 500
34.130 lib/ndkbuild/jni/AIOList.c File Reference . . . 500
34.130.1 Function Documentation . . . 500
intToString . . . 500
Deleteint . . . 500
TAIL_Q_LIST_IMPLEMENTATION . . . 500
TAIL_Q_LIST_IMPLEMENTATION . . . 500
intlistFirst . . . 500
Newintlist . . . 500
Deleteintlist . . . 500
intlistToString . . . 500
intlistSize . . . 500
intlistInsert . . . 500
34.130.2 Variable Documentation . . . 501
newone . . . 501
34.131 lib/AIOList.h File Reference . . . 501
34.131.1 Macro Definition Documentation . . . 501
TAIL_Q_LIST_TYPE . . . 501
TAIL_Q_LIST_ENTRY_TYPE . . . 501
TAIL_Q_LIST . . . 501
TAIL_Q_LIST_IMPLEMENTATION . . . 502
foreach_int . . . 502
foreach_CStringArray_p . . . 502
34.131.2 Function Documentation . . . 502
TAIL_Q_LIST . . . 502
TAIL_Q_LIST . . . 502

TAIL_Q_LIST_TYPE . . . 502
TAIL_Q_LIST_ENTRY_TYPE . . . 502
Newintlist . . . 502
Deleteintlist . . . 502
intlistToString . . . 502
intlistSize . . . 502
intlistFirst . . . 502
intlistInsert . . . 502
34.132 Lib/ndkbuild/jni/AIOList.h File Reference . . . 502
34.132.1 Macro Definition Documentation . . . 503
TAIL_Q_LIST_TYPE . . . 503
TAIL_Q_LIST_ENTRY_TYPE . . . 503
TAIL_Q_LIST . . . 503
TAIL_Q_LIST_IMPLEMENTATION . . . 503
foreach_int . . . 503
foreach_CStringArray_p . . . 503
34.132.2 Function Documentation . . . 503
TAIL_Q_LIST . . . 503
TAIL_Q_LIST . . . 503
TAIL_Q_LIST_TYPE . . . 503
TAIL_Q_LIST_ENTRY_TYPE . . . 503
Newintlist . . . 503
Deleteintlist . . . 503
intlistToString . . . 503
intlistSize . . . 503
intlistFirst . . . 503
intlistInsert . . . 503
34.133 Lib/AIOProductTypes.c File Reference . . . 504
34.133.1 Function Documentation . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
AIO_PRODUCT_CONSTANT . . . 504
NewAIOProductRange . . . 504
DeleteAIOProductRange . . . 504
AIOProductRangeStart . . . 505
AIOProductRangeEnd . . . 505
NewAIOProductGroup . . . 505
DeleteAIOProductGroup . . . 505
AIOProductGroupContains . . . 505
groupcpy . . . 505
34.134 Lib/ndkbuild/jni/AIOProductTypes.c File Reference . . . 505
34.134.1 Function Documentation . . . 505
AIO_PRODUCT_CONSTANT . . . 505
AIO_PRODUCT_CONSTANT . . . 505

AIO_PRODUCT_CONSTANT . . . 505
AIO_PRODUCT_CONSTANT . . . 505
AIO_PRODUCT_CONSTANT . . . 505
AIO_PRODUCT_CONSTANT . . . 505
AIO_PRODUCT_CONSTANT . . . 506
AIO_PRODUCT_CONSTANT . . . 506
NewAIOProductRange . . . 506
DeleteAIOProductRange . . . 506
AIOProductRangeStart . . . 506
AIOProductRangeEnd . . . 506
NewAIOProductGroup . . . 506
DeleteAIOProductGroup . . . 506
AIOProductGroupContains . . . 506
groupcpy . . . 506
34.135 lib/AIOProductTypes.h File Reference . . . 506
34.135.1 Macro Definition Documentation . . . 507
NUMARGS . . . 507
AIO_RANGE . . . 507
AIO_PRODUCT_GROUP . . . 507
AIO_PRODUCT_CONSTANT . . . 507
AIO_PRODUCT_EXTERN . . . 507
34.135.2 Typedef Documentation . . . 507
AIOProductRange . . . 507
AIOProductGroup . . . 507
34.135.3 Function Documentation . . . 507
NewAIOProductRange . . . 507
DeleteAIOProductRange . . . 507
AIOProductRangeStart . . . 507
AIOProductRangeEnd . . . 507
NewAIOProductGroup . . . 507
DeleteAIOProductGroup . . . 507
AIOProductGroupContains . . . 507
groupcpy . . . 507
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
AIO_PRODUCT_EXTERN . . . 508
34.136 lib/ndkbuild/jni/AIOProductTypes.h File Reference . . . 508
34.136.1 Macro Definition Documentation . . . 509
NUMARGS . . . 509
AIO_RANGE . . . 509
AIO_PRODUCT_GROUP . . . 509
AIO_PRODUCT_CONSTANT . . . 509
AIO_PRODUCT_EXTERN . . . 509
34.136.2 Typedef Documentation . . . 509

AIOProductRange . . . 509
AIOProductGroup . . . 509
34.136.3 Function Documentation . . . 509
NewAIOProductRange . . . 509
DeleteAIOProductRange . . . 509
AIOProductRangeStart . . . 509
AIOProductRangeEnd . . . 509
NewAIOProductGroup . . . 509
DeleteAIOProductGroup . . . 509
AIOProductGroupContains . . . 509
groupcpy . . . 509
AIO_PRODUCT_EXTERN . . . 509
AIO_PRODUCT_EXTERN . . . 509
AIO_PRODUCT_EXTERN . . . 510
AIO_PRODUCT_EXTERN . . . 510
AIO_PRODUCT_EXTERN . . . 510
AIO_PRODUCT_EXTERN . . . 510
AIO_PRODUCT_EXTERN . . . 510
AIO_PRODUCT_EXTERN . . . 510
34.137 lib/AIOTuple.c File Reference . . . 510
34.138 lib/ndkbuild/jni/AIOTuple.c File Reference . . . 510
34.139 lib/AIOTuple.h File Reference . . . 510
34.139.1 Macro Definition Documentation . . . 510
AIOTUPLE2_TYPE . . . 510
AIO_CHAR_ARRAY . . . 511
AIOTUPLE2_PTR . . . 511
AIOTUPLE2 . . . 511
AIOTUPLE2_TO_STR . . . 511
34.139.2 Function Documentation . . . 511
AIOTUPLE2_TYPE . . . 511
AIOTUPLE2_TYPE . . . 511
AIOTuple2_AIORET_TYPE__CStringArray_pToString . . . 511
DeleteAIOTuple2_AIORET_TYPE__CStringArray_p . . . 511
34.140 lib/ndkbuild/jni/AIOTuple.h File Reference . . . 511
34.140.1 Macro Definition Documentation . . . 511
AIOTUPLE2_TYPE . . . 511
AIO_CHAR_ARRAY . . . 511
AIOTUPLE2_PTR . . . 511
AIOTUPLE2 . . . 511
AIOTUPLE2_TO_STR . . . 511
34.140.2 Function Documentation . . . 511
AIOTUPLE2_TYPE . . . 511
AIOTUPLE2_TYPE . . . 512
AIOTuple2_AIORET_TYPE__CStringArray_pToString . . . 512
DeleteAIOTuple2_AIORET_TYPE__CStringArray_p . . . 512
34.141 lib/AIOTypes.h File Reference . . . 512
34.141.1 Macro Definition Documentation . . . 517
HAS_PTHREAD . . . 517
EXPORTED_FUNCTION . . . 517

CREATE_ENUM . . . 517
CREATE_ENUM_W_START . . . 517
LAST_ENUM . . . 518
FIRST_ENUM . . . 518
MIN_VALUE . . . 518
MAX_VALUE . . . 518
VALID_ENUM . . . 518
ERR_UNLESS_VALID_ENUM . . . 518
VALID_PRODUCT . . . 518
GCC_VERSION . . . 518
ACCES_DEPRECATED . . . 518
LAMBDA . . . 518
MIN . . . 518
MAX . . . 518
AUR_CBUF_SETUP . . . 518
AUR_CBUF_EXIT . . . 518
NUMBER_CHANNELS . . . 518
foreach_array . . . 518
AIO_MAKE_ERROR . . . 518
AIOUSB_ERROR_VALUE . . . 518
AIO_ASSERT . . . 518
AIO_ASSERT_RET . . . 518
AIO_ASSERT_AIORET_TYPE . . . 518
AIO_ASSERT_NO_RETURN . . . 518
AIO_ASSERT_EXIT . . . 518
AIO_ASSERT_ERR_NO_RETURN . . . 519
AIO_ASSERT_VALID_DATA . . . 519
AIO_ASSERT_USB . . . 519
AIO_ASSERT_DIOBUF . . . 519
AIO_ASSERT_CHANNELMASK . . . 519
AIO_ASSERT_CONFIG . . . 519
AIO_ASSERT_AIOCONTBUF . . . 519
AIO_ASSERT_AIOEITHER . . . 519
AIO_ERROR . . . 519
AIO_ERROR_VALID_DATA . . . 519
AIO_ERROR_VALID_DATA_RETVAL . . . 519
AIO_ERROR_VALID_AIORET_TYPE . . . 519
AIO_ERROR_AIOEITHER_VALID_DATA . . . 519
AIO_ERROR_VALID_DATA_W_CODE . . . 519
AIO_ERROR_VALID_DATA_WITH_CODE . . . 519
G_STMT_START . . . 519
G_STMT_END . . . 519
G_BREAKPOINT . . . 519
EXIT_FN_IF_NO_VALID_USB . . . 519
AIOUSB_ERROR_OFFSET . . . 519
LIBUSB_RESULT_TO_AIOUSB_RESULT . . . 519
AIOUSB_RESULT_TO_LIBUSB_RESULT . . . 519
ROOTCLOCK . . . 520
34.141.2 Typedef Documentation . . . 520

AIORET_TYPE . . . 520
AIORESULT . . . 520
COUNTS . . . 520
Ushort_Array . . . 520
AIOBufferType . . . 520
AIOUSB_BOOL . . . 520
AIO_NUMBER . . . 520
EnumStringLookup . . . 520
34.141.3 Enumeration Type Documentation . . . 520
AIO_SCAN_TYPE . . . 520
THREAD_STATUS . . . 520
AIOContinuousBufMode . . . 521
anonymous enum . . . 521
AIOUSB_BOOL_VAL . . . 521
ProductIDS . . . 521
anonymous enum . . . 525
DACRange . . . 526
FIFO_Method . . . 526
ResultCode . . . 526
anonymous enum . . . 529
ADRegister . . . 529
anonymous enum . . . 530
ADGainCode . . . 530
VENDOR_REQUEST . . . 531
anonymous enum . . . 533
ADCalMode . . . 533
AIOCommandCode . . . 533
34.142 /home/ndkbuild/jni/AIOTypes.h File Reference . . . 534
34.142.1 Macro Definition Documentation . . . 539
HAS_PTHREAD . . . 539
EXPORTED_FUNCTION . . . 539
CREATE_ENUM . . . 539
CREATE_ENUM_W_START . . . 539
LAST_ENUM . . . 540
FIRST_ENUM . . . 540
MIN_VALUE . . . 540
MAX_VALUE . . . 540
VALID_ENUM . . . 540
ERR_UNLESS_VALID_ENUM . . . 540
VALID_PRODUCT . . . 540
GCC_VERSION . . . 540
ACCES_DEPRECATED . . . 540
LAMBDA . . . 540
MIN . . . 540
MAX . . . 540
AUR_CBUF_SETUP . . . 540
AUR_CBUF_EXIT . . . 540
NUMBER_CHANNELS . . . 540
foreach_array . . . 540

AIO_MAKE_ERROR . . . 540
AIOUSB_ERROR_VALUE . . . 540
AIO_ASSERT . . . 540
AIO_ASSERT_RET . . . 540
AIO_ASSERT_AIORET_TYPE . . . 540
AIO_ASSERT_NO_RETURN . . . 540
AIO_ASSERT_EXIT . . . 540
AIO_ASSERT_ERR_NO_RETURN . . . 541
AIO_ASSERT_VALID_DATA . . . 541
AIO_ASSERT_USB . . . 541
AIO_ASSERT_DIOBUF . . . 541
AIO_ASSERT_CHANNELMASK . . . 541
AIO_ASSERT_CONFIG . . . 541
AIO_ASSERT_AIOCONTBUF . . . 541
AIO_ASSERT_AIOEITHER . . . 541
AIO_ERROR . . . 541
AIO_ERROR_VALID_DATA . . . 541
AIO_ERROR_VALID_DATA_RETVAL . . . 541
AIO_ERROR_VALID_AIORET_TYPE . . . 541
AIO_ERROR_AIOEITHER_VALID_DATA . . . 541
AIO_ERROR_VALID_DATA_W_CODE . . . 541
AIO_ERROR_VALID_DATA_WITH_CODE . . . 541
G_STMT_START . . . 541
G_STMT_END . . . 541
G_BREAKPOINT . . . 541
EXIT_FN_IF_NO_VALID_USB . . . 541
AIOUSB_ERROR_OFFSET . . . 541
LIBUSB_RESULT_TO_AIOUSB_RESULT . . . 541
AIOUSB_RESULT_TO_LIBUSB_RESULT . . . 541
ROOTCLOCK . . . 542
34.142.2 Typedef Documentation . . . 542
AIORET_TYPE . . . 542
AIORESULT . . . 542
COUNTS . . . 542
Ushort_Array . . . 542
AIOBufferType . . . 542
AIOUSB_BOOL . . . 542
AIO_NUMBER . . . 542
EnumStringLookup . . . 542
34.142.3 Enumeration Type Documentation . . . 542
AIO_SCAN_TYPE . . . 542
THREAD_STATUS . . . 542
AIOContinuousBufMode . . . 543
anonymous enum . . . 543
AIOUSB_BOOL_VAL . . . 543
ProductIDS . . . 543
anonymous enum . . . 547
DACRange . . . 548
FIFO_Method . . . 548

ResultCode . . . 548
anonymous enum . . . 551
ADRegister . . . 551
anonymous enum . . . 552
ADGainCode . . . 552
VENDOR_REQUEST . . . 553
anonymous enum . . . 555
ADCalMode . . . 555
AIOCommandCode . . . 555
34.143 lib/aiousb.h File Reference . . . 556
34.144 lib/ndkbuild/jni/aiousb.h File Reference . . . 556
34.145 lib/AIOUSB_ADC.c File Reference . . . 556
34.145.1 Macro Definition Documentation . . . 558
DEVICE_SAMPLE_BUFFER_SIZE . . . 558
STREAMING_PNA_DEFINITIONS . . . 558
34.145.2 Function Documentation . . . 559
adc_get_bulk_data . . . 559
ADC_ResetDevice . . . 559
ADC_GetConfigSize . . . 559
ADC_GetConfigRegisters . . . 559
ADC_ReadADConfigBlock . . . 559
ReadConfigBlock . . . 559
WriteConfigBlock . . . 559
ADC_Acquire_Reference_Counts . . . 559
AIOUSB_GetScan . . . 559
AIOUSB_ArrayCountsToVolts . . . 560
AIOUSB_ArrayVoltsToCounts . . . 560
AIOUSB_VoltsToCounts . . . 560
ADC_GetChannelV . . . 560
ADC_GetScanV . . . 561
ADC_GetScan . . . 561
ADC_GetConfig . . . 561
adcblock_valid_trigger_settings . . . 562
adcblock_valid_channel_settings . . . 562
valid_config_block . . . 562
adcblock_valid_size . . . 562
ADC_SetConfig . . . 562
ADC_CopyConfig . . . 562
ADC_RangeAll . . . 562
ADC_Range1 . . . 562
ADC_ADMode . . . 562
ADC_SetOversample . . . 563
ADC_GetOversample . . . 563
ADC_SetAllGainCodeAndDiffMode . . . 563
ADC_GetMaxClockRate . . . 563
ADC_ClockRateForADCProduct . . . 563
ADC_SetScanLimits . . . 563
ADC_SetCal . . . 563
ADC_CanCalibrate . . . 564

ADC_QueryCal . . . 564
ADC_Initialize . . . 564
ADC_BulkAcquire . . . 564
CreateSmartBuffer . . . 564
ADC_BulkPoll . . . 565
ADC_CreateFastITConfig . . . 565
ADC_GetADConfigBlock_Registers . . . 565
ADC_ClearFastITConfig . . . 565
ADC_CreateADBuf . . . 565
ADC_ClearADBuf . . . 565
ADC_InitFastITScanV . . . 565
ADC_ResetFastITScanV . . . 565
ADC_SetFastITScanVChannels . . . 565
ADC_Debug_Register_Settings . . . 565
ADConfigBlockToYAML . . . 565
ADC_GetFastITScanV . . . 566
ADC_GetITScanV . . . 566
AIOUSB_IsDiscardFirstSample . . . 566
AIOUSB_SetDiscardFirstSample . . . 566
AIOUSB_Copy_Config_Block . . . 566
AIOUSB_Validate_ADC_Device . . . 566
GetHiRef . . . 566
DoLoadCalTable . . . 567
AIOUSB_SetRangeSingle . . . 567
ConfigureAndBulkAcquire . . . 567
AIOUSB_ADC_InternalCal . . . 567
AIOUSB_SetRegister . . . 567
AIOUSB_GetRegister . . . 567
AIOUSB_SetAllGainCodeAndDiffMode . . . 567
AIOUSB_GetGainCode . . . 567
AIOUSB_SetGainCode . . . 567
AIOUSB_IsDifferentialMode . . . 567
AIOUSB_SetDifferentialMode . . . 568
AIOUSB_GetCalMode . . . 568
AIOUSB_SetCalMode . . . 568
AIOUSB_GetTriggerMode . . . 568
AIOUSB_SetTriggerMode . . . 568
AIOUSB_GetStartChannel . . . 568
AIOUSB_GetEndChannel . . . 568
AIOUSB_SetScanRange . . . 568
AIOUSB_GetOversample . . . 568
AIOUSB_SetOversample . . . 568
AIOUSB_ADC_ExternalCal . . . 568
34.145.3 Variable Documentation . . . 569
adRanges . . . 569
dRef . . . 569
34.146 /home/ndkbuild/jni/AIOUSB_ADC.c File Reference . . . 570
34.146.1 Macro Definition Documentation . . . 572
DEVICE_SAMPLE_BUFFER_SIZE . . . 572

STREAMING_PNA_DEFINITIONS . . . 572
34.146.2 Function Documentation . . . 572
adc_get_bulk_data . . . 572
ADC_ResetDevice . . . 572
ADC_GetConfigSize . . . 572
ADC_GetConfigRegisters . . . 572
ADC_ReadADConfigBlock . . . 572
ReadConfigBlock . . . 572
WriteConfigBlock . . . 572
ADC_Acquire_Reference_Counts . . . 572
AIOUSB_GetScan . . . 572
AIOUSB_ArrayCountsToVolts . . . 573
AIOUSB_ArrayVoltsToCounts . . . 573
AIOUSB_VoltsToCounts . . . 573
ADC_GetChannelV . . . 574
ADC_GetScanV . . . 574
ADC_GetScan . . . 574
ADC_GetConfig . . . 575
adcblock_valid_trigger_settings . . . 575
adcblock_valid_channel_settings . . . 575
valid_config_block . . . 575
adcblock_valid_size . . . 575
ADC_SetConfig . . . 575
ADC_CopyConfig . . . 575
ADC_RangeAll . . . 575
ADC_Range1 . . . 576
ADC_ADMode . . . 577
ADC_SetOversample . . . 577
ADC_GetOversample . . . 577
ADC_SetAllGainCodeAndDiffMode . . . 577
ADC_GetMaxClockRate . . . 577
ADC_ClockRateForADCProduct . . . 577
ADC_SetScanLimits . . . 577
ADC_SetCal . . . 578
ADC_CanCalibrate . . . 578
ADC_QueryCal . . . 578
ADC_Initialize . . . 578
ADC_BulkAcquire . . . 578
CreateSmartBuffer . . . 579
ADC_BulkPoll . . . 579
ADC_CreateFastITConfig . . . 579
ADC_GetADConfigBlock_Registers . . . 579
ADC_ClearFastITConfig . . . 579
ADC_CreateADBuf . . . 579
ADC_ClearADBuf . . . 579
ADC_InitFastITScanV . . . 579
ADC_ResetFastITScanV . . . 579
ADC_SetFastITScanVChannels . . . 580
ADC_Debug_Register_Settings . . . 580

ADConfigBlockToYAML . . . 580
ADC_GetFastITScanV . . . 580
ADC_GetITScanV . . . 580
AIOUSB_IsDiscardFirstSample . . . 580
AIOUSB_SetDiscardFirstSample . . . 580
AIOUSB_Copy_Config_Block . . . 581
AIOUSB_Validate_ADC_Device . . . 581
GetHiRef . . . 581
DoLoadCalTable . . . 581
AIOUSB_SetRangeSingle . . . 581
ConfigureAndBulkAcquire . . . 581
AIOUSB_ADC_InternalCal . . . 581
AIOUSB_SetRegister . . . 581
AIOUSB_GetRegister . . . 581
AIOUSB_SetAllGainCodeAndDiffMode . . . 581
AIOUSB_GetGainCode . . . 582
AIOUSB_SetGainCode . . . 582
AIOUSB_IsDifferentialMode . . . 583
AIOUSB_SetDifferentialMode . . . 583
AIOUSB_GetCalMode . . . 583
AIOUSB_SetCalMode . . . 583
AIOUSB_GetTriggerMode . . . 583
AIOUSB_SetTriggerMode . . . 583
AIOUSB_GetStartChannel . . . 583
AIOUSB_GetEndChannel . . . 583
AIOUSB_SetScanRange . . . 583
AIOUSB_GetOversample . . . 583
AIOUSB_SetOversample . . . 583
AIOUSB_ADC_ExternalCal . . . 583
34.146.3 Variable Documentation . . . 584
adRanges . . . 584
dRef . . . 584
34.147 /home/b/AIOUSB_ADC.h File Reference . . . 585
34.147.1 Function Documentation . . . 586
ADC_GetScanV . . . 586
ADC_RangeAll . . . 586
ADC_GetChannelV . . . 587
ADC_GetScan . . . 588
AIOUSB_GetScan . . . 588
ADC_GetConfig . . . 589
ADC_SetConfig . . . 589
ADC_Range1 . . . 590
ADC_ADMode . . . 590
ADC_SetScanLimits . . . 590
ADC_GetMaxClockRate . . . 590
ADC_ClockRateForADCProduct . . . 590
ADC_SetCal . . . 590
ADC_QueryCal . . . 590
ADC_CanCalibrate . . . 591

ADC_Initialize . . . 591
ADC_BulkAcquire . . . 591
ADC_BulkPoll . . . 591
ADC_InitFastITScanV . . . 591
ADC_CreateFastITConfig . . . 591
ADC_ResetFastITScanV . . . 592
ADC_SetFastITScanVChannels . . . 592
ADC_GetFastITScanV . . . 592
ADC_GetITScanV . . . 592
ADC_GetOversample . . . 592
ADC_SetOversample . . . 592
WriteConfigBlock . . . 592
ReadConfigBlock . . . 592
AIOUSB_SetAllGainCodeAndDiffMode . . . 593
AIOUSB_GetGainCode . . . 593
AIOUSB_SetGainCode . . . 593
AIOUSB_IsDifferentialMode . . . 593
AIOUSB_ADC_ExternalCal . . . 593
ADC_SetAllGainCodeAndDiffMode . . . 594
AIOUSB_SetDifferentialMode . . . 595
AIOUSB_GetCalMode . . . 596
AIOUSB_SetCalMode . . . 596
AIOUSB_SetOversample . . . 596
AIOUSB_GetOversample . . . 596
AIOUSB_GetTriggerMode . . . 596
AIOUSB_SetTriggerMode . . . 596
AIOUSB_GetStartChannel . . . 596
AIOUSB_GetEndChannel . . . 596
AIOUSB_SetScanRange . . . 596
AIOUSB_SetStreamingBlockSize . . . 596
AIOUSB_IsDiscardFirstSample . . . 596
AIOUSB_SetDiscardFirstSample . . . 596
CreateSmartBuffer . . . 597
AIOUSB_ADC_InternalCal . . . 598
ADC_GetADConfigBlock_Registers . . . 598
AIOUSB_SetRegister . . . 598
AIOUSB_GetRegister . . . 598
34.148 lib/ndkbuild/jni/AIOUSB_ADC.h File Reference . . . 598
34.148.1 Function Documentation . . . 600
ADC_GetScanV . . . 600
ADC_RangeAll . . . 600
ADC_GetChannelV . . . 600
ADC_GetScan . . . 601
AIOUSB_GetScan . . . 601
ADC_GetConfig . . . 602
ADC_SetConfig . . . 602
ADC_Range1 . . . 602
ADC_ADMode . . . 602
ADC_SetScanLimits . . . 603

ADC_GetMaxClockRate . . . 603
ADC_ClockRateForADCProduct . . . 603
ADC_SetCal . . . 603
ADC_QueryCal . . . 603
ADC_CanCalibrate . . . 603
ADC_Initialize . . . 603
ADC_BulkAcquire . . . 604
ADC_BulkPoll . . . 604
ADC_InitFastITScanV . . . 604
ADC_CreateFastITConfig . . . 604
ADC_ResetFastITScanV . . . 604
ADC_SetFastITScanVChannels . . . 604
ADC_GetFastITScanV . . . 604
ADC_GetITScanV . . . 605
ADC_GetOversample . . . 605
ADC_SetOversample . . . 605
WriteConfigBlock . . . 605
ReadConfigBlock . . . 605
AIOUSB_SetAllGainCodeAndDiffMode . . . 605
AIOUSB_GetGainCode . . . 605
AIOUSB_SetGainCode . . . 605
AIOUSB_IsDifferentialMode . . . 606
AIOUSB_ADC_ExternalCal . . . 606
ADC_SetAllGainCodeAndDiffMode . . . 607
AIOUSB_SetDifferentialMode . . . 607
AIOUSB_GetCalMode . . . 607
AIOUSB_SetCalMode . . . 607
AIOUSB_SetOversample . . . 608
AIOUSB_GetOversample . . . 608
AIOUSB_GetTriggerMode . . . 608
AIOUSB_SetTriggerMode . . . 608
AIOUSB_GetStartChannel . . . 608
AIOUSB_GetEndChannel . . . 608
AIOUSB_SetScanRange . . . 608
AIOUSB_SetStreamingBlockSize . . . 608
AIOUSB_IsDiscardFirstSample . . . 608
AIOUSB_SetDiscardFirstSample . . . 608
CreateSmartBuffer . . . 608
AIOUSB_ADC_InternalCal . . . 609
ADC_GetADConfigBlock_Registers . . . 609
AIOUSB_SetRegister . . . 609
AIOUSB_GetRegister . . . 609
34.149 lib/AIOUSB_Core.c File Reference . . . 609
34.149.1 Macro Definition Documentation . . . 610
BACKTRACE_DEBUG . . . 610
AIOUSB_ENABLE_MUTEX . . . 610
34.149.2 Function Documentation . . . 610
AIOUSB_Lock . . . 610
AIOUSB_UnLock . . . 610

AIOUSB_ResetChip . . . 610
AIOUSB_Validate_Lock . . . 610
AIOUSB_Validate . . . 610
ResolveDeviceIndex . . . 610
DeviceTableAtIndex . . . 611
DeviceTableAtIndex_Lock . . . 611
AIOUSB_GetDevice . . . 611
AIOUSB_GetConfigBlock . . . 611
AIOUSB_GetStreamingBlockSize . . . 611
AIOUSB_SetStreamingBlockSize . . . 611
AIOUSB_ClearFIFO . . . 611
AIOUSB_GetVersion . . . 611
AIOUSB_GetVersionDate . . . 611
AIOUSB_GetMiscClock . . . 611
AIOUSB_SetMiscClock . . . 611
AIOUSB_GetCommTimeout . . . 611
AIOUSB_SetCommTimeout . . . 611
AIOUSB_Validate_Device . . . 611
AIOUSB_InitConfigBlock . . . 611
AIOUSB_ADC_LoadCalTable . . . 611
AIOUSB_ADC_SetCalTable . . . 611
GenericVendorWrite . . . 612
GenericVendorRead . . . 612
34.149.3 Variable Documentation . . . 612
aio_errno . . . 612
productIDNameTable . . . 612
34.150 lib/ndkbuild/jni/AIOUSB_Core.c File Reference . . . 612
34.150.1 Macro Definition Documentation . . . 613
BACKTRACE_DEBUG . . . 613
AIOUSB_ENABLE_MUTEX . . . 613
34.150.2 Function Documentation . . . 613
AIOUSB_Lock . . . 613
AIOUSB_UnLock . . . 613
AIOUSB_ResetChip . . . 613
AIOUSB_Validate_Lock . . . 613
AIOUSB_Validate . . . 613
ResolveDeviceIndex . . . 613
DeviceTableAtIndex . . . 613
DeviceTableAtIndex_Lock . . . 614
AIOUSB_GetDevice . . . 614
AIOUSB_GetConfigBlock . . . 614
AIOUSB_GetStreamingBlockSize . . . 614
AIOUSB_SetStreamingBlockSize . . . 614
AIOUSB_ClearFIFO . . . 614
AIOUSB_GetVersion . . . 614
AIOUSB_GetVersionDate . . . 614
AIOUSB_GetMiscClock . . . 614
AIOUSB_SetMiscClock . . . 614
AIOUSB_GetCommTimeout . . . 614

AIOUSB_SetCommTimeout . . . 614
AIOUSB_Validate_Device . . . 614
AIOUSB_InitConfigBlock . . . 614
AIOUSB_ADC_LoadCalTable . . . 614
AIOUSB_ADC_SetCalTable . . . 614
GenericVendorWrite . . . 615
GenericVendorRead . . . 615
34.150.3 Variable Documentation . . . 615
aio_errno . . . 615
productIDNameTable . . . 615
34.151 lib/AIOUSB_Core.h File Reference . . . 615
34.151.1 Macro Definition Documentation . . . 616
PUBLIC_EXTERN . . . 616
PRIVATE . . . 616
PROD_NAME_SIZE . . . 616
34.151.2 Typedef Documentation . . . 616
AIOOption . . . 616
AIOError . . . 616
34.151.3 Function Documentation . . . 616
ADC_ResetDevice . . . 616
AIOUSB_GetDeviceSerialNumber . . . 616
AIOUSB_ResetChip . . . 616
AIOUSB_Lock . . . 616
AIOUSB_UnLock . . . 617
AIOUSB_InitTest . . . 617
AIOUSB_Validate . . . 617
AIOUSB_Validate_Lock . . . 617
DeviceTableAtIndex . . . 617
DeviceTableAtIndex_Lock . . . 617
AIOUSB_GetDevice . . . 617
AIOUSB_GetConfigBlock . . . 617
AIOUSB_SetMiscClock . . . 617
AIOUSB_GetMiscClock . . . 617
AIOUSB_SetCommTimeout . . . 617
AIOUSB_GetCommTimeout . . . 617
AIOUSB_GetVersion . . . 617
AIOUSB_GetVersionDate . . . 617
AIOUSB_GetResultCodeAsString . . . 617
AIOUSB_VoltsToCounts . . . 617
AIOUSB_ADC_LoadCalTable . . . 617
AIOUSB_ADC_SetCalTable . . . 618
AIOUSB_ClearFIFO . . . 618
AIOUSB_GetStreamingBlockSize . . . 618
AIOUSB_InitConfigBlock . . . 618
GenericVendorRead . . . 618
GenericVendorWrite . . . 618
AIOUSB_Validate_Device . . . 618
34.151.4 Variable Documentation . . . 618
aio_errno . . . 618

adRanges . . . 618
AIOUSB_INIT_PATTERN . . . 618
aiousbInit . . . 619
34.152 lib/ndkbuild/jni/AIOUSB_Core.h File Reference . . . 619
34.152.1 Macro Definition Documentation . . . 620
PUBLIC_EXTERN . . . 620
PRIVATE . . . 620
PROD_NAME_SIZE . . . 620
34.152.2 Typedef Documentation . . . 620
AIOOption . . . 620
AIOError . . . 620
34.152.3 Function Documentation . . . 620
ADC_ResetDevice . . . 620
AIOUSB_GetDeviceSerialNumber . . . 620
AIOUSB_ResetChip . . . 620
AIOUSB_Lock . . . 620
AIOUSB_UnLock . . . 620
AIOUSB_InitTest . . . 620
AIOUSB_Validate . . . 620
AIOUSB_Validate_Lock . . . 620
DeviceTableAtIndex . . . 621
DeviceTableAtIndex_Lock . . . 621
AIOUSB_GetDevice . . . 621
AIOUSB_GetConfigBlock . . . 621
AIOUSB_SetMiscClock . . . 621
AIOUSB_GetMiscClock . . . 621
AIOUSB_SetCommTimeout . . . 621
AIOUSB_GetCommTimeout . . . 621
AIOUSB_GetVersion . . . 621
AIOUSB_GetVersionDate . . . 621
AIOUSB_GetResultCodeAsString . . . 621
AIOUSB_VoltsToCounts . . . 621
AIOUSB_ADC_LoadCalTable . . . 621
AIOUSB_ADC_SetCalTable . . . 621
AIOUSB_ClearFIFO . . . 622
AIOUSB_GetStreamingBlockSize . . . 622
AIOUSB_InitConfigBlock . . . 623
GenericVendorRead . . . 623
GenericVendorWrite . . . 623
AIOUSB_Validate_Device . . . 623
34.152.4 Variable Documentation . . . 623
aio_errno . . . 623
adRanges . . . 623
AIOUSB_INIT_PATTERN . . . 623
aiousbInit . . . 623
34.153 lib/AIOUSB_CTR.c File Reference . . . 623
34.153.1 Macro Definition Documentation . . . 624
RETURN_IF_INVALID_INPUT . . . 624
JUMP_IF_INVALID_INPUT . . . 624

JUMP_IF_NO_VALID_USB . . . 624
34.153.2 Function Documentation . . . 624
CTR_8254Mode . . . 624
CTR_8254Load . . . 624
CTR_8254ModeLoad . . . 624
CTR_8254ReadModeLoad . . . 624
CTR_8254Read . . . 625
CTR_8254ReadAll . . . 625
CTR_8254ReadStatus . . . 625
CTR_CalculateCountersForClock . . . 625
CTR_StartOutputFreq . . . 625
CTR_8254SelectGate . . . 625
CTR_8254ReadLatched . . . 625
34.154 Lib/ndkbuild/jni/AIOUSB_CTR.c File Reference . . . 625
34.154.1 Macro Definition Documentation . . . 626
RETURN_IF_INVALID_INPUT . . . 626
JUMP_IF_INVALID_INPUT . . . 626
JUMP_IF_NO_VALID_USB . . . 626
34.154.2 Function Documentation . . . 626
CTR_8254Mode . . . 626
CTR_8254Load . . . 626
CTR_8254ModeLoad . . . 626
CTR_8254ReadModeLoad . . . 626
CTR_8254Read . . . 626
CTR_8254ReadAll . . . 626
CTR_8254ReadStatus . . . 626
CTR_CalculateCountersForClock . . . 626
CTR_StartOutputFreq . . . 627
CTR_8254SelectGate . . . 627
CTR_8254ReadLatched . . . 627
34.155 Lib/AIOUSB_CTR.h File Reference . . . 627
34.155.1 Function Documentation . . . 627
CTR_CalculateCountersForClock . . . 627
CTR_8254Mode . . . 627
CTR_8254Load . . . 627
CTR_8254ModeLoad . . . 628
CTR_8254ReadModeLoad . . . 628
CTR_8254Read . . . 628
CTR_8254ReadAll . . . 628
CTR_8254ReadStatus . . . 628
CTR_StartOutputFreq . . . 628
CTR_8254SelectGate . . . 628
CTR_8254ReadLatched . . . 628
34.156 Lib/ndkbuild/jni/AIOUSB_CTR.h File Reference . . . 628
34.156.1 Function Documentation . . . 628
CTR_CalculateCountersForClock . . . 628
CTR_8254Mode . . . 629
CTR_8254Load . . . 629
CTR_8254ModeLoad . . . 629

CTR_8254ReadModeLoad . . . 629
CTR_8254Read . . . 629
CTR_8254ReadAll . . . 629
CTR_8254ReadStatus . . . 629
CTR_StartOutputFreq . . . 629
CTR_8254SelectGate . . . 629
CTR_8254ReadLatched . . . 629
34.157 lib/AIOUSB_CustomEEPROM.c File Reference . . . 629
34.157.1 Macro Definition Documentation . . . 629
EXIT_FN_IF_NO_VALID_USB . . . 629
34.157.2 Function Documentation . . . 630
CustomEEPROMWrite . . . 630
CustomEEPROMRead . . . 630
34.158 lib/ndkbuild/jni/AIOUSB_CustomEEPROM.c File Reference . . . 630
34.158.1 Macro Definition Documentation . . . 630
EXIT_FN_IF_NO_VALID_USB . . . 630
34.158.2 Function Documentation . . . 630
CustomEEPROMWrite . . . 630
CustomEEPROMRead . . . 631
34.159 lib/AIOUSB_CustomEEPROM.h File Reference . . . 631
34.159.1 Function Documentation . . . 631
CustomEEPROMWrite . . . 631
CustomEEPROMRead . . . 631
34.160 lib/ndkbuild/jni/AIOUSB_CustomEEPROM.h File Reference . . . 631
34.160.1 Function Documentation . . . 631
CustomEEPROMWrite . . . 632
CustomEEPROMRead . . . 632
34.161 lib/AIOUSB_DAC.c File Reference . . . 632
34.161.1 Function Documentation . . . 632
DACDirect . . . 632
DACMultiDirect . . . 632
DACSetBoardRange . . . 633
DACOutputOpen . . . 633
DACOutputClose . . . 633
DACOutputCloseNoEnd . . . 633
DACOutputSetCount . . . 633
DACOutputFrame . . . 633
DACOutputFrameRaw . . . 633
DACOutputStart . . . 633
DACOutputSetInterlock . . . 633
34.162 lib/ndkbuild/jni/AIOUSB_DAC.c File Reference . . . 633
34.162.1 Function Documentation . . . 634
DACDirect . . . 634
DACMultiDirect . . . 634
DACSetBoardRange . . . 635
DACOutputOpen . . . 635
DACOutputClose . . . 635
DACOutputCloseNoEnd . . . 635
DACOutputSetCount . . . 635

DACOutputFrame . . . 635
DACOutputFrameRaw . . . 635
DACOutputStart . . . 635
DACOutputSetInterlock . . . 635
34.163 lib/AIOUSB_DAC.h File Reference . . . 635
34.163.1 Function Documentation . . . 635
DACDirect . . . 635
DACMultiDirect . . . 635
DACSetBoardRange . . . 636
DACOutputOpen . . . 636
DACOutputClose . . . 636
DACOutputCloseNoEnd . . . 636
DACOutputSetCount . . . 636
DACOutputFrame . . . 636
DACOutputFrameRaw . . . 636
DACOutputStart . . . 636
DACOutputSetInterlock . . . 636
34.164 lib/ndkbuild/jni/AIOUSB_DAC.h File Reference . . . 636
34.164.1 Function Documentation . . . 637
DACDirect . . . 637
DACMultiDirect . . . 637
DACSetBoardRange . . . 638
DACOutputOpen . . . 638
DACOutputClose . . . 638
DACOutputCloseNoEnd . . . 638
DACOutputSetCount . . . 638
DACOutputFrame . . . 638
DACOutputFrameRaw . . . 638
DACOutputStart . . . 638
DACOutputSetInterlock . . . 638
34.165 lib/AIOUSB_DIO.c File Reference . . . 638
34.165.1 Macro Definition Documentation . . . 638
GET_ENDPOINT . . . 638
34.165.2 Function Documentation . . . 638
MASK_BYTES_SIZE . . . 639
TRISTATE_BYTES_SIZE . . . 639
aiousb_htons . . . 639
DIO_ConfigureWithDIOBuf . . . 639
DIO_Configure . . . 639
DIO_ConfigureEx . . . 639
DIO_ConfigurationQuery . . . 639
DIO_WriteAll . . . 639
DIO_Write8 . . . 639
DIO_Write1 . . . 639
DIO_ReadAll . . . 639
DIO_ReadIntoDIOBuf . . . 639
DIO_ReadAllToDIOBuf . . . 639
DIO_ReadAllToCharStr . . . 639
DIO_Read8 . . . 639

DIO_Read1 . . . 639
DIO_StreamOpen . . . 639
DIO_StreamClose . . . 639
DIO_StreamSetClocks . . . 639
pow_of_minsize . . . 639
DIO_StreamFrame . . . 639
34.166 /home/b/ndkbuild/jni/AIOUSB_DIO.c File Reference . . . 640
34.166.1 Macro Definition Documentation . . . 640
GET_ENDPOINT . . . 640
34.166.2 Function Documentation . . . 640
MASK_BYTES_SIZE . . . 640
TRISTATE_BYTES_SIZE . . . 640
aiousb_htons . . . 640
DIO_ConfigureWithDIOBuf . . . 641
DIO_Configure . . . 641
DIO_ConfigureEx . . . 641
DIO_ConfigurationQuery . . . 641
DIO_WriteAll . . . 641
DIO_Write8 . . . 641
DIO_Write1 . . . 641
DIO_ReadAll . . . 641
DIO_ReadIntoDIOBuf . . . 641
DIO_ReadAllToDIOBuf . . . 641
DIO_ReadAllToCharStr . . . 641
DIO_Read8 . . . 641
DIO_Read1 . . . 641
DIO_StreamOpen . . . 641
DIO_StreamClose . . . 641
DIO_StreamSetClocks . . . 641
pow_of_minsize . . . 641
DIO_StreamFrame . . . 641
34.167 /home/b/AIOUSB_DIO.h File Reference . . . 642
34.167.1 Function Documentation . . . 642
DIO_ConfigureWithDIOBuf . . . 642
DIO_Configure . . . 642
DIO_ConfigureEx . . . 642
DIO_ConfigurationQuery . . . 642
DIO_WriteAll . . . 642
DIO_Write8 . . . 642
DIO_Write1 . . . 642
DIO_ReadAllToDIOBuf . . . 642
DIO_ReadIntoDIOBuf . . . 642
DIO_ReadAll . . . 644
DIO_ReadAllToCharStr . . . 644
DIO_Read8 . . . 644
DIO_Read1 . . . 644
DIO_StreamOpen . . . 644
DIO_StreamClose . . . 644
DIO_StreamSetClocks . . . 644

DIO_StreamFrame . . . 644
34.168 lib/ndkbuild/jni/AIOUSB_DIO.h File Reference . . . 644
34.168.1 Function Documentation . . . 645
DIO_ConfigureWithDIOBuf . . . 645
DIO_Configure . . . 645
DIO_ConfigureEx . . . 645
DIO_ConfigurationQuery . . . 645
DIO_WriteAll . . . 645
DIO_Write8 . . . 645
DIO_Write1 . . . 645
DIO_ReadAllToDIOBuf . . . 645
DIO_ReadIntoDIOBuf . . . 645
DIO_ReadAll . . . 645
DIO_ReadAllToCharStr . . . 645
DIO_Read8 . . . 645
DIO_Read1 . . . 645
DIO_StreamOpen . . . 645
DIO_StreamClose . . . 645
DIO_StreamSetClocks . . . 645
DIO_StreamFrame . . . 646
34.169 lib/AIOUSB_Log.c File Reference . . . 646
34.169.1 Variable Documentation . . . 646
cont_thread . . . 646
message_lock . . . 646
outfile . . . 646
AIOUSB_DEBUG_LEVEL . . . 646
34.170 lib/ndkbuild/jni/AIOUSB_Log.c File Reference . . . 646
34.170.1 Variable Documentation . . . 647
cont_thread . . . 647
message_lock . . . 647
outfile . . . 647
AIOUSB_DEBUG_LEVEL . . . 647
34.171 lib/AIOUSB_Log.h File Reference . . . 647
34.171.1 Macro Definition Documentation . . . 647
GREEN . . . 647
RED . . . 647
MAGENTA . . . 647
CYAN . . . 647
AIO_DEVEL_STR . . . 648
AIO_DEBUG_STR . . . 648
AIO_WARN_STR . . . 648
AIO_INFO_STR . . . 648
AIO_ERROR_STR . . . 648
AIO_FATAL_STR . . . 648
AIO_RESET_STR . . . 648
AIOUSB_LOG . . . 648
AIOUSB_DEVEL . . . 648
AIOUSB_DEBUG . . . 648
AIOUSB_WARN . . . 648

AIOUSB_INFO . . . 648
AIOUSB_ERROR . . . 648
AIOUSB_FATAL . . . 648
34.171.2 Enumeration Type Documentation . . . 648
AIO_DEBUG_LEVEL . . . 648
34.171.3 Variable Documentation . . . 648
AIOUSB_DEBUG_LEVEL . . . 648
LOG_LEVEL . . . 648
cont_thread . . . 649
message_lock . . . 649
outfile . . . 649
34.172 lib/ndkbuild/jni/AIOUSB_Log.h File Reference . . . 649
34.172.1 Macro Definition Documentation . . . 649
GREEN . . . 649
RED . . . 649
MAGENTA . . . 649
CYAN . . . 649
AIO_DEVEL_STR . . . 649
AIO_DEBUG_STR . . . 649
AIO_WARN_STR . . . 650
AIO_INFO_STR . . . 650
AIO_ERROR_STR . . . 650
AIO_FATAL_STR . . . 650
AIO_RESET_STR . . . 650
AIOUSB_LOG . . . 650
AIOUSB_DEVEL . . . 650
AIOUSB_DEBUG . . . 650
AIOUSB_WARN . . . 650
AIOUSB_INFO . . . 650
AIOUSB_ERROR . . . 650
AIOUSB_FATAL . . . 650
34.172.2 Enumeration Type Documentation . . . 650
AIO_DEBUG_LEVEL . . . 650
34.172.3 Variable Documentation . . . 650
AIOUSB_DEBUG_LEVEL . . . 650
LOG_LEVEL . . . 650
cont_thread . . . 651
message_lock . . . 651
outfile . . . 651
34.173 lib/AIOUSB_Properties.c File Reference . . . 651
34.173.1 Macro Definition Documentation . . . 651
RESULT_TEXT_SIZE . . . 651
NUM_RESULT_CODES . . . 651
34.173.2 Function Documentation . . . 651
non_usb_supported_device . . . 651
AIOUSB_GetDeviceByProductID . . . 651
AIOUSB_GetDeviceSerialNumber . . . 652
GetDeviceSerialNumber . . . 652
GetDeviceBySerialNumber . . . 652

AIOUSB_FindDevices . . . 652
AIOUSB_FindDeviceIndicesByGroup . . . 652
AIOUSB_FindDevicesByGroup . . . 652
AIOUSB_GetDeviceProperties . . . 652
AIOUSB_GetResultCodeAsString . . . 652
AIOUSB_ShowDevices . . . 652
AIOUSB_ListDevices . . . 652
34.174 lib/ndkbuild/jni/AIOUSB_Properties.c File Reference . . . 652
34.174.1 Macro Definition Documentation . . . 653
RESULT_TEXT_SIZE . . . 653
NUM_RESULT_CODES . . . 653
34.174.2 Function Documentation . . . 653
non_usb_supported_device . . . 653
AIOUSB_GetDeviceByProductID . . . 653
AIOUSB_GetDeviceSerialNumber . . . 653
GetDeviceSerialNumber . . . 653
GetDeviceBySerialNumber . . . 654
AIOUSB_FindDevices . . . 654
AIOUSB_FindDeviceIndicesByGroup . . . 654
AIOUSB_FindDevicesByGroup . . . 654
AIOUSB_GetDeviceProperties . . . 654
AIOUSB_GetResultCodeAsString . . . 654
AIOUSB_ShowDevices . . . 654
AIOUSB_ListDevices . . . 654
34.175 lib/AIOUSB_Properties.h File Reference . . . 654
34.175.1 Enumeration Type Documentation . . . 655
AIODisplayType . . . 655
34.175.2 Function Documentation . . . 655
AIOUSB_GetDeviceByProductID . . . 655
GetDeviceBySerialNumber . . . 655
GetDeviceSerialNumber . . . 655
AIOUSB_GetDeviceSerialNumber . . . 655
AIOUSB_GetDeviceProperties . . . 655
AIOUSB_GetResultCodeAsString . . . 655
AIOUSB_ListDevices . . . 656
AIOUSB_ShowDevices . . . 656
AIOUSB_FindDevices . . . 656
AIOUSB_FindDevicesByGroup . . . 656
AIOUSB_FindDeviceIndicesByGroup . . . 656
34.176 lib/ndkbuild/jni/AIOUSB_Properties.h File Reference . . . 656
34.176.1 Enumeration Type Documentation . . . 656
AIODisplayType . . . 656
34.176.2 Function Documentation . . . 657
AIOUSB_GetDeviceByProductID . . . 657
GetDeviceBySerialNumber . . . 657
GetDeviceSerialNumber . . . 657
AIOUSB_GetDeviceSerialNumber . . . 657
AIOUSB_GetDeviceProperties . . . 657
AIOUSB_GetResultCodeAsString . . . 657

AIOUSB_ListDevices . . . 657
AIOUSB_ShowDevices . . . 657
AIOUSB_FindDevices . . . 657
AIOUSB_FindDevicesByGroup . . . 658
AIOUSB_FindDeviceIndicesByGroup . . . 658
34.177 lib/AIOUSB_USB.c File Reference . . . 658
34.178 lib/ndkbuild/jni/AIOUSB_USB.c File Reference . . . 658
34.179 lib/AIOUSB_USB.h File Reference . . . 658
34.179.1 Macro Definition Documentation . . . 658
usb_control_xfer . . . 658
usb_bulk_xfer . . . 658
usb_open . . . 658
usb_close . . . 658
usb_free_devices . . . 658
usb_get_devices . . . 658
34.180 lib/ndkbuild/jni/AIOUSB_USB.h File Reference . . . 658
34.180.1 Macro Definition Documentation . . . 659
usb_control_xfer . . . 659
usb_bulk_xfer . . . 659
usb_open . . . 659
usb_close . . . 659
usb_free_devices . . . 659
usb_get_devices . . . 659
34.181 lib/AIOUSB_WDG.c File Reference . . . 659
34.181.1 Function Documentation . . . 659
NewWDGConfig . . . 659
doSomething . . . 659
DeleteWDGConfig . . . 659
WDG_SetConfig . . . 659
WDG_GetStatus . . . 660
WDG_Pet . . . 661
34.182 lib/ndkbuild/jni/AIOUSB_WDG.c File Reference . . . 661
34.182.1 Function Documentation . . . 661
NewWDGConfig . . . 661
doSomething . . . 661
DeleteWDGConfig . . . 661
WDG_SetConfig . . . 661
WDG_GetStatus . . . 662
WDG_Pet . . . 662
34.183 lib/AIOUSB_WDG.h File Reference . . . 662
34.183.1 Enumeration Type Documentation . . . 663
WDGVals . . . 663
34.183.2 Function Documentation . . . 663
NewWDGConfig . . . 663
DeleteWDGConfig . . . 663
WDG_SetConfig . . . 663
WDG_GetStatus . . . 663
WDG_Pet . . . 663
34.184 lib/ndkbuild/jni/AIOUSB_WDG.h File Reference . . . 664

34.184.1 Enumeration Type Documentation . . . 664
WDGVals . . . 664
34.184.2 Function Documentation . . . 664
NewWDGConfig . . . 664
DeleteWDGConfig . . . 664
WDG_SetConfig . . . 664
WDG_GetStatus . . . 665
WDG_Pet . . . 665
34.185 lib/AIOUSBDevice.c File Reference . . . 665
34.185.1 Function Documentation . . . 666
NewAIOUSBDevice . . . 666
NewAIOUSBDeviceFromJSON . . . 666
AIOUSBDeviceToJSON . . . 666
DeleteAIOUSBDevice . . . 666
AIOUSBDeviceInitializeWithProductID . . . 666
AIOUSBDeviceSetTimeout . . . 666
AIOUSBDeviceGetTimeout . . . 666
AIOUSBDeviceCopyADCConfigBlock . . . 666
AIOUSBDeviceSetADCConfigBlock . . . 666
AIOUSBDeviceSize . . . 666
AIOUSBDeviceGetADCConfigBlock . . . 666
AIOUSBDeviceWriteADCConfig . . . 666
AIOUSBDeviceSetTesting . . . 666
AIOUSBDeviceGetStreamingBlockSize . . . 666
AIOUSBDeviceGetTesting . . . 666
AIOUSBDeviceGetUSBHandle . . . 666
AIOUSBDeviceSetUSBHandle . . . 666
AIOUSBDeviceGetUSBHandleFromDeviceIndex . . . 666
AIOUSBDeviceGetDiscardFirstSample . . . 666
AIOUSBDeviceSetDiscardFirstSample . . . 666
34.186 lib/ndkbuild/jni/AIOUSBDevice.c File Reference . . . 666
34.186.1 Function Documentation . . . 667
NewAIOUSBDevice . . . 667
NewAIOUSBDeviceFromJSON . . . 667
AIOUSBDeviceToJSON . . . 667
DeleteAIOUSBDevice . . . 667
AIOUSBDeviceInitializeWithProductID . . . 667
AIOUSBDeviceSetTimeout . . . 667
AIOUSBDeviceGetTimeout . . . 667
AIOUSBDeviceCopyADCConfigBlock . . . 667
AIOUSBDeviceSetADCConfigBlock . . . 667
AIOUSBDeviceSize . . . 667
AIOUSBDeviceGetADCConfigBlock . . . 667
AIOUSBDeviceWriteADCConfig . . . 667
AIOUSBDeviceSetTesting . . . 667
AIOUSBDeviceGetStreamingBlockSize . . . 667
AIOUSBDeviceGetTesting . . . 667
AIOUSBDeviceGetUSBHandle . . . 667
AIOUSBDeviceSetUSBHandle . . . 667

AIOUSBDeviceGetUSBHandleFromDeviceIndex . . . 667
AIOUSBDeviceGetDiscardFirstSample . . . 667
AIOUSBDeviceSetDiscardFirstSample . . . 667
34.187 /home/b/AIOUSBDevice.h File Reference . . . 667
34.187.1 Typedef Documentation . . . 668
DeviceDescriptor . . . 668
34.187.2 Function Documentation . . . 668
AIOUSBDeviceToJSON . . . 668
NewAIOUSBDeviceFromJSON . . . 668
AIOUSBDeviceInitializeWithProductID . . . 668
AIOUSBDeviceGetUSBHandle . . . 668
AIOUSBDeviceGetUSBHandleFromDeviceIndex . . . 668
AIOUSBDeviceSetUSBHandle . . . 668
AIOUSBDeviceSetADCConfigBlock . . . 668
AIOUSBDeviceGetADCConfigBlock . . . 668
AIOUSBDeviceCopyADCConfigBlock . . . 668
AIOUSBDeviceSetTesting . . . 668
AIOUSBDeviceSize . . . 668
AIOUSBDeviceGetTesting . . . 668
AIOUSBDeviceGetStreamingBlockSize . . . 668
AIOUSBDeviceGetDiscardFirstSample . . . 668
AIOUSBDeviceSetDiscardFirstSample . . . 668
AIOUSBDeviceSetTimeout . . . 668
AIOUSBDeviceGetTimeout . . . 668
AIOUSBDeviceWriteADCConfig . . . 669
34.188 /home/b/ndkbuild/jni/AIOUSBDevice.h File Reference . . . 669
34.188.1 Typedef Documentation . . . 669
DeviceDescriptor . . . 669
34.188.2 Function Documentation . . . 669
AIOUSBDeviceToJSON . . . 669
NewAIOUSBDeviceFromJSON . . . 669
AIOUSBDeviceInitializeWithProductID . . . 669
AIOUSBDeviceGetUSBHandle . . . 669
AIOUSBDeviceGetUSBHandleFromDeviceIndex . . . 669
AIOUSBDeviceSetUSBHandle . . . 669
AIOUSBDeviceSetADCConfigBlock . . . 669
AIOUSBDeviceGetADCConfigBlock . . . 670
AIOUSBDeviceCopyADCConfigBlock . . . 670
AIOUSBDeviceSetTesting . . . 670
AIOUSBDeviceSize . . . 670
AIOUSBDeviceGetTesting . . . 670
AIOUSBDeviceGetStreamingBlockSize . . . 670
AIOUSBDeviceGetDiscardFirstSample . . . 670
AIOUSBDeviceSetDiscardFirstSample . . . 670
AIOUSBDeviceSetTimeout . . . 670
AIOUSBDeviceGetTimeout . . . 670
AIOUSBDeviceWriteADCConfig . . . 670
34.189 /home/b/cJSON.c File Reference . . . 670
34.189.1 Function Documentation . . . 671

cJSON_GetErrorPtr . . . 671
cJSON_InitHooks . . . 671
cJSON_Delete . . . 671
cJSON_ParseWithOpts . . . 671
cJSON_Parse . . . 671
cJSON_Print . . . 671
cJSON_PrintUnformatted . . . 671
cJSON_AsInteger . . . 671
cJSON_GetArraySize . . . 671
cJSON_GetArrayItem . . . 671
cJSON_GetObjectItem . . . 671
cJSON_AddItemToArray . . . 671
cJSON_AddItemToObject . . . 671
cJSON_AddItemReferenceToArray . . . 671
cJSON_AddItemReferenceToObject . . . 671
cJSON_DetachItemFromArray . . . 671
cJSON_DeleteItemFromArray . . . 671
cJSON_DetachItemFromObject . . . 671
cJSON_DeleteItemFromObject . . . 671
cJSON_ReplaceItemInArray . . . 671
cJSON_ReplaceItemInObject . . . 671
cJSON_CreateNull . . . 671
cJSON_CreateTrue . . . 671
cJSON_CreateFalse . . . 671
cJSON_CreateBool . . . 671
cJSON_CreateNumber . . . 671
cJSON_CreateString . . . 671
cJSON_CreateArray . . . 671
cJSON_CreateObject . . . 671
cJSON_CreateIntArray . . . 671
cJSON_CreateFloatArray . . . 671
cJSON_CreateDoubleArray . . . 672
cJSON_CreateStringArray . . . 672
cJSON_Duplicate . . . 672
cJSON_Minify . . . 672
34.190 lib/ndkbuild/jni/cJSON.c File Reference . . . 672
34.190.1 Function Documentation . . . 673
cJSON_GetErrorPtr . . . 673
cJSON_InitHooks . . . 673
cJSON_Delete . . . 673
cJSON_ParseWithOpts . . . 673
cJSON_Parse . . . 673
cJSON_Print . . . 673
cJSON_PrintUnformatted . . . 673
cJSON_AsInteger . . . 673
cJSON_GetArraySize . . . 673
cJSON_GetArrayItem . . . 673
cJSON_GetObjectItem . . . 673
cJSON_AddItemToArray . . . 673

cJSON_AddItemToObject . . . 673
cJSON_AddItemReferenceToArray . . . 673
cJSON_AddItemReferenceToObject . . . 673
cJSON_DetachItemFromArray . . . 673
cJSON_DeleteItemFromArray . . . 673
cJSON_DetachItemFromObject . . . 673
cJSON_DeleteItemFromObject . . . 673
cJSON_ReplaceItemInArray . . . 673
cJSON_ReplaceItemInObject . . . 673
cJSON_CreateNull . . . 673
cJSON_CreateTrue . . . 673
cJSON_CreateFalse . . . 673
cJSON_CreateBool . . . 673
cJSON_CreateNumber . . . 673
cJSON_CreateString . . . 673
cJSON_CreateArray . . . 673
cJSON_CreateObject . . . 673
cJSON_CreateIntArray . . . 673
cJSON_CreateFloatArray . . . 673
cJSON_CreateDoubleArray . . . 673
cJSON_CreateStringArray . . . 673
cJSON_Duplicate . . . 673
cJSON_Minify . . . 674
34.191 lib/cJSON.h File Reference . . . 674
34.191.1 Macro Definition Documentation . . . 675
cJSON_False . . . 675
cJSON_True . . . 675
cJSON_NULL . . . 675
cJSON_Number . . . 675
cJSON_String . . . 675
cJSON_Array . . . 675
cJSON_Object . . . 675
cJSON_IsReference . . . 675
cJSON_AddNullToObject . . . 675
cJSON_AddTrueToObject . . . 675
cJSON_AddFalseToObject . . . 675
cJSON_AddBoolToObject . . . 675
cJSON_AddNumberToObject . . . 675
cJSON_AddStringToObject . . . 675
cJSON_SetIntValue . . . 675
34.191.2 Typedef Documentation . . . 675
cJSON . . . 675
cJSON_Hooks . . . 675
34.191.3 Function Documentation . . . 675
cJSON_InitHooks . . . 675
cJSON_Parse . . . 675
cJSON_Print . . . 675
cJSON_PrintUnformatted . . . 675
cJSON_Delete . . . 675

cJSON_AsInteger . . . 675
cJSON_GetArraySize . . . 675
cJSON_GetArrayItem . . . 675
cJSON_GetObjectItem . . . 675
cJSON_GetErrorPtr . . . 676
cJSON_CreateNull . . . 676
cJSON_CreateTrue . . . 676
cJSON_CreateFalse . . . 676
cJSON_CreateBool . . . 676
cJSON_CreateNumber . . . 676
cJSON_CreateString . . . 676
cJSON_CreateArray . . . 676
cJSON_CreateObject . . . 676
cJSON_CreateIntArray . . . 676
cJSON_CreateFloatArray . . . 676
cJSON_CreateDoubleArray . . . 676
cJSON_CreateStringArray . . . 676
cJSON_AddItemToArray . . . 676
cJSON_AddItemToObject . . . 676
cJSON_AddItemReferenceToArray . . . 676
cJSON_AddItemReferenceToObject . . . 676
cJSON_DetachItemFromArray . . . 676
cJSON_DeleteItemFromArray . . . 676
cJSON_DetachItemFromObject . . . 676
cJSON_DeleteItemFromObject . . . 676
cJSON_ReplaceItemInArray . . . 676
cJSON_ReplaceItemInObject . . . 676
cJSON_Duplicate . . . 676
cJSON_ParseWithOpts . . . 676
cJSON_Minify . . . 676
34.192 b/ndkbuild/jni/cJSON.h File Reference . . . 676
34.192.1 Macro Definition Documentation . . . 678
cJSON_False . . . 678
cJSON_True . . . 678
cJSON_NULL . . . 678
cJSON_Number . . . 678
cJSON_String . . . 678
cJSON_Array . . . 678
cJSON_Object . . . 678
cJSON_IsReference . . . 678
cJSON_AddNullToObject . . . 678
cJSON_AddTrueToObject . . . 678
cJSON_AddFalseToObject . . . 678
cJSON_AddBoolToObject . . . 678
cJSON_AddNumberToObject . . . 678
cJSON_AddStringToObject . . . 678
cJSON_SetIntValue . . . 678
34.192.2 Typedef Documentation . . . 678
cJSON . . . 678

cJSON_Hooks . . . 678
34.192.3 Function Documentation . . . 678
cJSON_InitHooks . . . 678
cJSON_Parse . . . 678
cJSON_Print . . . 678
cJSON_PrintUnformatted . . . 678
cJSON_Delete . . . 678
cJSON_AsInteger . . . 678
cJSON_GetArraySize . . . 678
cJSON_GetArrayItem . . . 678
cJSON_GetObjectItem . . . 678
cJSON_GetErrorPtr . . . 678
cJSON_CreateNull . . . 678
cJSON_CreateTrue . . . 678
cJSON_CreateFalse . . . 678
cJSON_CreateBool . . . 678
cJSON_CreateNumber . . . 679
cJSON_CreateString . . . 679
cJSON_CreateArray . . . 679
cJSON_CreateObject . . . 679
cJSON_CreateIntArray . . . 679
cJSON_CreateFloatArray . . . 679
cJSON_CreateDoubleArray . . . 679
cJSON_CreateStringArray . . . 679
cJSON_AddItemToArray . . . 679
cJSON_AddItemToObject . . . 679
cJSON_AddItemReferenceToArray . . . 679
cJSON_AddItemReferenceToObject . . . 679
cJSON_DetachItemFromArray . . . 679
cJSON_DeleteItemFromArray . . . 679
cJSON_DetachItemFromObject . . . 679
cJSON_DeleteItemFromObject . . . 679
cJSON_ReplaceItemInArray . . . 679
cJSON_ReplaceItemInObject . . . 679
cJSON_Duplicate . . . 679
cJSON_ParseWithOpts . . . 679
cJSON_Minify . . . 679
34.193 lib/CStringArray.c File Reference . . . 679
34.193.1 Function Documentation . . . 679
NewCStringArrayWithStrings . . . 679
NewCStringArray . . . 679
NewCStringArrayFromCArgs . . . 680
DeleteCStringArray . . . 680
CopyCStringArray . . . 680
CStringArrayToString . . . 680
CStringArrayToStringWithDelimeter . . . 680
34.194 lib/ndkbuild/jni/CStringArray.c File Reference . . . 680
34.194.1 Function Documentation . . . 680
NewCStringArrayWithStrings . . . 680

NewCStringArray . . . 680
NewCStringArrayFromCArgs . . . 680
DeleteCStringArray . . . 680
CopyCStringArray . . . 680
CStringArrayToString . . . 680
CStringArrayToStringWithDelimeter . . . 680
34.195 Lib/CStringArray.h File Reference . . . 680
34.195.1 Macro Definition Documentation . . . 681
STRING_ARRAY . . . 681
34.195.2 Typedef Documentation . . . 681
CStringArray . . . 681
34.195.3 Function Documentation . . . 681
NewCStringArray . . . 681
NewCStringArrayWithStrings . . . 681
NewCStringArrayFromCArgs . . . 681
DeleteCStringArray . . . 681
CopyCStringArray . . . 681
CStringArrayToString . . . 681
CStringArrayToStringWithDelimeter . . . 681
34.196 Lib/ndkbuild/jni/CStringArray.h File Reference . . . 681
34.196.1 Macro Definition Documentation . . . 682
STRING_ARRAY . . . 682
34.196.2 Typedef Documentation . . . 682
CStringArray . . . 682
34.196.3 Function Documentation . . . 682
NewCStringArray . . . 682
NewCStringArrayWithStrings . . . 682
NewCStringArrayFromCArgs . . . 682
DeleteCStringArray . . . 682
CopyCStringArray . . . 682
CStringArrayToString . . . 682
CStringArrayToStringWithDelimeter . . . 682
34.197 Lib/DIOBuf.c File Reference . . . 682
34.197.1 Function Documentation . . . 682
NewDIOBuf . . . 682
NewDIOBufFromChar . . . 683
NewDIOBufFromBinStr . . . 683
DIOBufReplaceString . . . 683
DIOBufReplaceBinString . . . 683
DeleteDIOBuf . . . 683
DIOBufResize . . . 683
DIOBufSize . . . 683
DIOBufByteSize . . . 683
DIOBufToString . . . 683
DIOBufToHex . . . 684
DIOBufToBinary . . . 684
DIOBufToInvertedBinary . . . 684
DIOBufSetIndex . . . 684
DIOBufGetIndex . . . 684

DIOBufGetByteAtIndex . . . 684
DIOBufSetByteAtIndex . . . 684
34.198 lib/ndkbuild/jni/DIOBuf.c File Reference . . . 684
34.198.1 Function Documentation . . . 685
NewDIOBuf . . . 685
NewDIOBufFromChar . . . 685
NewDIOBufFromBinStr . . . 685
DIOBufReplaceString . . . 686
DIOBufReplaceBinString . . . 686
DeleteDIOBuf . . . 686
DIOBufResize . . . 686
DIOBufSize . . . 686
DIOBufByteSize . . . 686
DIOBufToString . . . 686
DIOBufToHex . . . 686
DIOBufToBinary . . . 686
DIOBufToInvertedBinary . . . 686
DIOBufSetIndex . . . 687
DIOBufGetIndex . . . 687
DIOBufGetByteAtIndex . . . 687
DIOBufSetByteAtIndex . . . 687
34.199 lib/DIOBuf.h File Reference . . . 687
34.199.1 Typedef Documentation . . . 688
DIOBufferType . . . 688
34.199.2 Function Documentation . . . 688
NewDIOBuf . . . 688
NewDIOBufFromChar . . . 688
NewDIOBufFromBinStr . . . 688
DeleteDIOBuf . . . 689
DIOBufReplaceString . . . 689
DIOBufReplaceBinString . . . 689
DIOBufToHex . . . 689
DIOBufToBinary . . . 689
DIOBufToInvertedBinary . . . 689
DIOBufResize . . . 689
DIOBufSize . . . 689
DIOBufByteSize . . . 689
DIOBufToString . . . 689
DIOBufSetIndex . . . 690
DIOBufGetIndex . . . 690
DIOBufGetByteAtIndex . . . 690
DIOBufSetByteAtIndex . . . 690
34.200 lib/ndkbuild/jni/DIOBuf.h File Reference . . . 690
34.200.1 Typedef Documentation . . . 691
DIOBufferType . . . 691
34.200.2 Function Documentation . . . 691
NewDIOBuf . . . 691
NewDIOBufFromChar . . . 691
NewDIOBufFromBinStr . . . 691

DeleteDIOBuf . . . 692
DIOBufReplaceString . . . 692
DIOBufReplaceBinString . . . 692
DIOBufToHex . . . 692
DIOBufToBinary . . . 692
DIOBufToInvertedBinary . . . 692
DIOBufResize . . . 692
DIOBufSize . . . 692
DIOBufByteSize . . . 692
DIOBufToString . . . 692
DIOBufSetIndex . . . 693
DIOBufGetIndex . . . 693
DIOBufGetByteAtIndex . . . 693
DIOBufSetByteAtIndex . . . 693
34.201 lib/mocks/mock_aiocontbuf_get_data.c File Reference . . . 693
34.201.1 Function Documentation . . . 693
aiocontbuf_get_bulk_data . . . 693
34.202 lib/mocks/mock_aiocontbuf_get_data_arduino.c File Reference . . . 694
34.202.1 Function Documentation . . . 694
aiocontbuf_get_bulk_data . . . 694
adc_get_bulk_data . . . 694
CloseAllDevices . . . 694
34.203 lib/mocks/mock_capture_usb.c File Reference . . . 694
34.203.1 Detailed Description . . . 695
34.203.2 Typedef Documentation . . . 695
init_device . . . 695
34.203.3 Enumeration Type Documentation . . . 695
IO_DIRECTION . . . 695
34.203.4 Function Documentation . . . 696
mock_usb_control_transfer . . . 696
mock_usb_bulk_transfer . . . 696
mock_usb_request . . . 696
mock_usb_reset_device . . . 696
mock_usb_put_config . . . 696
mock_usb_get_config . . . 696
InitializeUSBDevice . . . 696
34.203.5 Variable Documentation . . . 696
orig_usb_control_transfer . . . 696
orig_usb_bulk_transfer . . . 696
orig_usb_request . . . 696
orig_usb_reset_device . . . 696
orig_usb_put_config . . . 696
orig_usb_get_config . . . 696
outfile . . . 696
34.204 lib/mocks/mock_dio.c File Reference . . . 696
34.204.1 Enumeration Type Documentation . . . 697
IO_DIRECTION . . . 697
34.204.2 Function Documentation . . . 697
save_results . . . 697

mock_usb_control_transfer . . . 697
mock_usb_bulk_transfer . . . 697
34.205 lib/mocks/mock_usb_xfers.c File Reference . . . 697
34.205.1 Typedef Documentation . . . 698
add_devices_fn . . . 698
34.205.2 Enumeration Type Documentation . . . 698
IO_DIRECTION . . . 698
34.205.3 Function Documentation . . . 698
mock_usb_control_transfer . . . 698
mock_usb_bulk_transfer . . . 698
mock_usb_reset_device . . . 698
mock_USBDevicePutADCConfigBlock . . . 698
mock_USBDeviceFetchADCConfigBlock . . . 698
AddAllACCESUSBDevices . . . 698
34.205.4 Variable Documentation . . . 698
direction . . . 698
KEEP . . . 698
34.206 lib/ndkbuild/jni/libaiousb.h File Reference . . . 698
34.206.1 Macro Definition Documentation . . . 708
MATLAB . . . 708
LENGTH_AD_GAIN_CODE_STRINGS . . . 708
34.206.2 Typedef Documentation . . . 708
AIORESULT . . . 708
AIORET_TYPE . . . 708
AIOChannelRange . . . 708
AIOCountsConverter . . . 708
34.206.3 Function Documentation . . . 708
NewDIOBuf . . . 708
NewDIOBufFromChar . . . 709
NewDIOBufFromBinStr . . . 709
DeleteDIOBuf . . . 709
DIOBufReplaceString . . . 709
DIOBufReplaceBinString . . . 709
DIOBufToHex . . . 709
DIOBufToBinary . . . 710
DIOBufToInvertedBinary . . . 710
DIOBufResize . . . 711
DIOBufSize . . . 711
DIOBufByteSize . . . 711
DIOBufToString . . . 711
DIOBufSetIndex . . . 711
DIOBufGetIndex . . . 711
DIOBufGetByteAtIndex . . . 711
DIOBufSetByteAtIndex . . . 711
DIO_ConfigureWithDIOBuf . . . 711
DIO_Configure . . . 711
DIO_ConfigureEx . . . 711
DIO_ConfigurationQuery . . . 711
DIO_WriteAll . . . 711

DIO_Write8 . . . 712
DIO_Write1 . . . 712
DIO_ReadAllToDIOBuf . . . 712
DIO_ReadIntoDIOBuf . . . 712
DIO_ReadAll . . . 712
DIO_ReadAllToCharStr . . . 712
DIO_Read8 . . . 712
DIO_Read1 . . . 712
DIO_StreamOpen . . . 712
DIO_StreamClose . . . 712
DIO_StreamSetClocks . . . 712
DIO_StreamFrame . . . 712
ADC_ResetDevice . . . 713
AIOUSB_GetDeviceSerialNumber . . . 713
AIOUSB_ResetChip . . . 713
AIOUSB_Lock . . . 713
AIOUSB_UnLock . . . 713
AIOUSB_InitTest . . . 713
AIOUSB_Validate . . . 713
AIOUSB_Validate_Lock . . . 713
DeviceTableAtIndex . . . 713
DeviceTableAtIndex_Lock . . . 713
AIOUSB_GetDevice . . . 713
AIOUSB_GetConfigBlock . . . 713
AIOUSB_SetMiscClock . . . 713
AIOUSB_GetMiscClock . . . 713
AIOUSB_SetCommTimeout . . . 713
AIOUSB_GetCommTimeout . . . 713
AIOUSB_GetVersion . . . 713
AIOUSB_GetVersionDate . . . 713
AIOUSB_GetResultCodeAsString . . . 713
AIOUSB_VoltsToCounts . . . 714
AIOUSB_ADC_LoadCalTable . . . 715
AIOUSB_ADC_SetCalTable . . . 715
AIOUSB_ClearFIFO . . . 715
AIOUSB_GetStreamingBlockSize . . . 715
AIOUSB_InitConfigBlock . . . 715
GenericVendorRead . . . 715
GenericVendorWrite . . . 715
AIOUSB_Validate_Device . . . 716
NewAIOContinuousBuf . . . 716
NewAIOContinuousBufForCounts . . . 716
NewAIOContinuousBufForVolts . . . 716
DeleteAIOContinuousBuf . . . 716
AIOContinuousBufInitConfiguration . . . 716
AIOContinuousBufInitADCConfigBlock . . . 716
AIOContinuousBufGetCallback . . . 716
AIOContinuousBufSetCallback . . . 716
AIOContinuousBufSetStreamingBlockSize . . . 716

AIOContinuousBufGetStreamingBlockSize . . . 716
AIOContinuousBufGetADCConfigBlock . . . 716
AIOContinuousBufSetNumberChannels . . . 716
AIOContinuousBufGetNumberChannels . . . 717
AIOContinuousBufGetOversample . . . 717
AIOContinuousBufSetOversample . . . 717
AIOContinuousBufNumberChannels . . . 717
AIOContinuousBufSetBaseSize . . . 717
AIOContinuousBufGetBaseSize . . . 717
AIOContinuousBufGetBufferSize . . . 717
AIOContinuousBufSetUnitSize . . . 717
AIOContinuousBufGetUnitSize . . . 717
AIOContinuousBufSetTesting . . . 717
AIOContinuousBufGetTesting . . . 717
AIOContinuousBufSendPreConfig . . . 717
AIOContinuousBufSetStartAndEndChannel . . . 717
AIOContinuousBufSetAllGainCodeAndDiffMode . . . 717
AIOContinuousBufGetDeviceIndex . . . 717
AIOContinuousBufSetDiscardFirstSample . . . 717
AIOContinuousBufSetChannelMask . . . 717
AIOContinuousBufNumberSignals . . . 717
AIOContinuousBufSetChannelRange . . . 717
AIOContinuousBufSaveConfig . . . 717
AIOContinuousBufSetDeviceIndex . . . 717
AIOContinuousBufResetDevice . . . 717
AIOContinuousBufSetTimeout . . . 717
AIOContinuousBufGetTimeout . . . 717
AIOContinuousBufSetDebug . . . 717
AIOContinuousBufGetDebug . . . 717
AIOContinuousBufGetNumberScans . . . 717
AIOContinuousBufSetNumberScans . . . 717
AIOContinuousBufNumberWriteSamplesRemaining . . . 718
AIOContinuousBufNumberSamplesAvailable . . . 718
AIOContinuousBufGetNumberSamplesPerScan . . . 718
AIOContinuousBufGetTotalSamplesExpected . . . 718
AIOContinuousBufReset . . . 718
AIOContinuousBufPushN . . . 718
AIOContinuousBufPopN . . . 718
AIOContinuousBufLock . . . 718
AIOContinuousBufUnlock . . . 718
AIOContinuousBufCallbackStart . . . 718
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction . . . 718
AIOContinuousBufStopAcquisition . . . 719
AIOContinuousBufInitiateCallbackAcquisition . . . 719
AIOContinuousBufGetReadPosition . . . 719
AIOContinuousBufGetWritePosition . . . 719
AIOContinuousBufGetRemainingSize . . . 719
AIOContinuousBufGetStatus . . . 719
AIOContinuousBufGetExitCode . . . 719

AIOContinuousBufGetRunStatus . . . 719
AIOContinuousBufPending . . . 719
AIOContinuousBufGetScansRead . . . 719
AIOContinuousBufReadIntegerScanCounts . . . 719
AIOContinuousBufReadCompleteScanCounts . . . 719
AIOContinuousBufReadIntegerNumberOfScans . . . 719
AIOContinuousBufSetCountsBuffer . . . 720
AIOContinuousBufSetVoltsBuffer . . . 720
AIOContinuousBufCountScansAvailable . . . 720
AIOContinuousBufSetClock . . . 720
AIOContinuousBufGetClock . . . 720
AIOContinuousBufEnd . . . 720
AIOContinuousBufSimpleSetupConfig . . . 720
AIOContinuousBufRead . . . 720
AIOContinuousBufWrite . . . 720
AIOContinuousBufWriteCounts . . . 720
AIOContinuousBufCleanup . . . 720
AIOContinuousBufToJSON . . . 720
NewAIOContinuousBufFromJSON . . . 720
CTR_CalculateCountersForClock . . . 720
CTR_8254Mode . . . 721
CTR_8254Load . . . 721
CTR_8254ModeLoad . . . 721
CTR_8254ReadModeLoad . . . 721
CTR_8254Read . . . 721
CTR_8254ReadAll . . . 721
CTR_8254ReadStatus . . . 721
CTR_StartOutputFreq . . . 721
CTR_8254SelectGate . . . 721
CTR_8254ReadLatched . . . 721
ADCConfigBlockInit . . . 721
ADCConfigBlockInitForCounterScan . . . 721
ADCConfigBlockInitializeDefault . . . 721
ADC_VerifyAndCorrectConfigBlock . . . 721
ADCConfigBlockSetAllGainCodeAndDiffMode . . . 721
ADCConfigBlockSetRegister . . . 721
ADCConfigBlockGetGainCode . . . 721
ADCConfigBlockSetGainCode . . . 721
ADCConfigBlockSetClockRate . . . 722
ADCConfigBlockGetClockRate . . . 722
ADCConfigBlockSetScanRange . . . 722
ADCConfigBlockSetStartChannel . . . 722
ADCConfigBlockSetEndChannel . . . 722
ADCConfigBlockSetChannelRange . . . 722
ADCConfigBlockSetCalMode . . . 722
ADCConfigBlockGetCalMode . . . 722
ADCConfigBlockToYAML . . . 722
ADCConfigBlockGetStartChannel . . . 722
ADCConfigBlockGetEndChannel . . . 722

ADCConfigBlockGetOversample . . . 722
ADCConfigBlockSetOversample . . . 722
ADCConfigBlockGetTimeout . . . 722
ADCConfigBlockSetTimeout . . . 722
ADCConfigBlockGetTriggerMode . . . 722
ADCConfigBlockSetTriggerMode . . . 722
ADCConfigBlockSetReference . . . 723
ADCConfigBlockSetTriggerEdge . . . 723
ADCConfigBlockSetDifferentialMode . . . 723
ADCConfigBlockSetRangeSingle . . . 723
ADCConfigBlockCopy . . . 723
ADCConfigBlockSetDevice . . . 723
ADCConfigBlockSetAIOUSBDevice . . . 723
ADCConfigBlockGetAIOUSBDevice . . . 723
ADCConfigBlockInitializeFromAIOUSBDevice . . . 723
ADCConfigBlockSetTesting . . . 723
ADCConfigBlockGetTesting . . . 723
ADCConfigBlockSetSize . . . 723
ADCConfigBlockGetSize . . . 723
ADCConfigBlockSetDebug . . . 723
ADCConfigBlockGetDebug . . . 723
ADCConfigBlockToJSON . . . 723
NewADCConfigBlockFromJSON . . . 723
DeleteADCConfigBlock . . . 723
is_all_digits . . . 723
NewAIOChannelMask . . . 723
DeleteAIOChannelMask . . . 723
NewAIOChannelMaskFromStr . . . 723
NewAIOChannelMaskFromChr . . . 724
AIOChannelMaskToString . . . 724
AIOChannelMaskToStringAtIndex . . . 724
AIOChannelMaskGetMask . . . 724
AIOChannelMaskGetMaskAtIndex . . . 724
AIOChannelMaskNumberChannels . . . 724
AIOChannelMaskNumberSignals . . . 725
AIOChannelMaskGetSize . . . 725
AIOChannelMaskIndices . . . 725
AIOChannelMaskNextIndex . . . 725
AIOChannelMaskSetMaskFromInt . . . 725
AIOChannelMaskSetMaskAtIndex . . . 725
AIOChannelMaskSetMaskFromStr . . . 725
NewAIOChannelRangeFromStr . . . 725
DeleteAIOChannelRange . . . 725
AIOChannelRangeToStr . . . 725
AIOChannelRangeGetStart . . . 725
AIOChannelRangeGetEnd . . . 725
AIOChannelRangeGetGain . . . 725
NewAIOBuf . . . 725
DeleteAIOBuf . . . 725

AIOBufGetSize . . . 725
AIOBufGetType . . . 725
AIOBufGetRaw . . . 725
AIOBufRead . . . 725
AIOBufWrite . . . 725
AIOBufGetIterator . . . 726
AIOBufIteratorGetValue . . . 726
AIOBufIteratorIsValid . . . 726
AIOBufIteratorNext . . . 726
NewAIOCountsConverterWithBuffer . . . 726
NewAIOCountsConverter . . . 726
NewAIOCountsConverterFromAIOContinuousBuf . . . 726
NewAIOCountsConverterWithScanLimiter . . . 726
AIOCountsConverterReset . . . 726
DeleteAIOCountsConverter . . . 726
AIOCountsConverterConvertNScans . . . 726
AIOCountsConverterConvertAllAvailableScans . . . 726
AIOCountsConverterConvert . . . 726
AIOCountsConverterConvertFifo . . . 726
NewAIOGainRangeFromADCConfigBlock . . . 727
DeleteAIOGainRange . . . 727
NewAIODeviceInfo . . . 727
DeleteAIODeviceInfo . . . 727
AIODeviceInfoGetName . . . 727
AIODeviceInfoGet . . . 727
AIODeviceInfoGetCounters . . . 727
AIODeviceInfoGetDIOBytes . . . 727
NewAIODeviceQuery . . . 727
DeleteAIODeviceQuery . . . 727
AIODeviceQueryToStr . . . 727
AIODeviceQueryToRepr . . . 727
AIODeviceQueryGetProductID . . . 727
AIODeviceQueryNameSize . . . 728
AIODeviceQueryGetName . . . 728
AIODeviceQueryGetNumDIOBytes . . . 728
AIODeviceQueryGetNumCounters . . . 728
AIODeviceQueryGetIndex . . . 728
AIODeviceTableAddDeviceToDeviceTable . . . 729
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice . . . 729
AIODeviceTablePopulateTable . . . 729
AIODeviceTablePopulateTableTest . . . 729
AIODeviceTableClearDevices . . . 729
ClearDevices . . . 729
AIODeviceTableGetDeviceAtIndex . . . 729
AIODeviceTableGetAIOUSBDeviceAtIndex . . . 729
AIODeviceTableGetUSBDeviceAtIndex . . . 729
QueryDeviceInfo . . . 729
GetDevices . . . 729
GetSafeDeviceName . . . 730

ProductIDToName . . . 730
ProductNameToID . . . 730
AIOUSB_Init . . . 731
AIOUSB_EnsureOpen . . . 731
AIOUSB_IsInit . . . 731
AIOUSB_Exit . . . 731
AIOUSB_Reset . . . 731
AIODeviceTableInit . . . 731
ClearAIODeviceTable . . . 731
CloseAllDevices . . . 731
AIOUSB_GetAllDevices . . . 731
AIOUSBGetError . . . 731
NewAIOFifo . . . 731
DeleteAIOFifo . . . 731
AIOFifoReset . . . 731
AIOFifoRead . . . 731
AIOFifoWrite . . . 731
AIOFifoWriteAllOrNone . . . 731
AIOFifoReadAllOrNone . . . 731
AIOFifoGetRefSize . . . 731
NewAIOFifoTYPE . . . 731
Push . . . 731
PushN . . . 731
PopN . . . 732
AIOFifoWriteSizeRemaining . . . 732
AIOFifoWriteSizeRemainingNumElements . . . 732
AIOFifoReadSize . . . 732
AIOFifoReadSizeNumElements . . . 732
AIOFifoGetSize . . . 732
AIOFifoGetSizeNumElements . . . 732
AIOFifoResize . . . 732
AIOFifoReadPosition . . . 732
AIOFifoWritePosition . . . 732
AIOEitherClear . . . 732
AIOEitherSetRight . . . 732
AIOEitherGetRight . . . 732
AIOEitherSetLeft . . . 732
AIOEitherGetLeft . . . 732
AIOEitherHasError . . . 732
AIOEitherToString . . . 732
AIOEitherToInt . . . 732
AIOEitherToShort . . . 732
AIOEitherToUnsigned . . . 732
AIOEitherToDouble . . . 732
AIOEitherToAIONumber . . . 732
AIOEitherToAIORetType . . . 732
ADC_GetScanV . . . 732
ADC_RangeAll . . . 733
ADC_GetChannelV . . . 733

ADC_GetScan . . . 733
AIOUSB_GetScan . . . 734
ADC_GetConfig . . . 734
ADC_SetConfig . . . 735
ADC_Range1 . . . 735
ADC_ADMode . . . 735
ADC_SetScanLimits . . . 735
ADC_GetMaxClockRate . . . 735
ADC_ClockRateForADCProduct . . . 736
ADC_SetCal . . . 736
ADC_QueryCal . . . 736
ADC_CanCalibrate . . . 736
ADC_Initialize . . . 736
ADC_BulkAcquire . . . 736
ADC_BulkPoll . . . 737
ADC_InitFastITScanV . . . 737
ADC_CreateFastITConfig . . . 737
ADC_ResetFastITScanV . . . 737
ADC_SetFastITScanVChannels . . . 737
ADC_GetFastITScanV . . . 737
ADC_GetITScanV . . . 737
ADC_GetOversample . . . 737
ADC_SetOversample . . . 738
WriteConfigBlock . . . 738
ReadConfigBlock . . . 738
AIOUSB_SetAllGainCodeAndDiffMode . . . 738
AIOUSB_GetGainCode . . . 738
AIOUSB_SetGainCode . . . 738
AIOUSB_IsDifferentialMode . . . 738
AIOUSB_ADC_ExternalCal . . . 738
ADC_SetAllGainCodeAndDiffMode . . . 740
AIOUSB_SetDifferentialMode . . . 740
AIOUSB_GetCalMode . . . 740
AIOUSB_SetCalMode . . . 740
AIOUSB_SetOversample . . . 740
AIOUSB_GetOversample . . . 740
AIOUSB_GetTriggerMode . . . 740
AIOUSB_SetTriggerMode . . . 741
AIOUSB_GetStartChannel . . . 741
AIOUSB_GetEndChannel . . . 741
AIOUSB_SetScanRange . . . 741
AIOUSB_SetStreamingBlockSize . . . 741
AIOUSB_IsDiscardFirstSample . . . 741
AIOUSB_SetDiscardFirstSample . . . 741
CreateSmartBuffer . . . 741
AIOUSB_ADC_InternalCal . . . 741
ADC_GetADConfigBlock_Registers . . . 742
AIOUSB_SetRegister . . . 742
AIOUSB_GetRegister . . . 742

DACDirect . . . 742
DACMultiDirect . . . 742
DACSetBoardRange . . . 743
DACOutputOpen . . . 743
DACOutputClose . . . 743
DACOutputCloseNoEnd . . . 743
DACOutputSetCount . . . 743
DACOutputFrame . . . 743
DACOutputFrameRaw . . . 743
DACOutputStart . . . 743
DACOutputSetInterlock . . . 743
CustomEEPROMWrite . . . 743
CustomEEPROMRead . . . 743
AIOUSBDeviceToJSON . . . 743
NewAIOUSBDeviceFromJSON . . . 743
AIOUSBDeviceInitializeWithProductID . . . 743
AIOUSBDeviceGetUSBHandle . . . 743
AIOUSBDeviceGetUSBHandleFromDeviceIndex . . . 743
AIOUSBDeviceSetUSBHandle . . . 743
AIOUSBDeviceSetADCConfigBlock . . . 743
AIOUSBDeviceGetADCConfigBlock . . . 743
AIOUSBDeviceCopyADCConfigBlock . . . 743
AIOUSBDeviceSetTesting . . . 743
AIOUSBDeviceSize . . . 743
AIOUSBDeviceGetTesting . . . 743
AIOUSBDeviceGetStreamingBlockSize . . . 743
AIOUSBDeviceGetDiscardFirstSample . . . 744
AIOUSBDeviceSetDiscardFirstSample . . . 744
AIOUSBDeviceSetTimeout . . . 744
AIOUSBDeviceGetTimeout . . . 744
AIOUSBDeviceWriteADCConfig . . . 744
AIOUSB_GetDeviceByProductID . . . 744
GetDeviceBySerialNumber . . . 744
GetDeviceSerialNumber . . . 744
AIOUSB_GetDeviceProperties . . . 744
AIOUSB_ListDevices . . . 744
AIOUSB_ShowDevices . . . 744
AIOUSB_FindDevices . . . 744
AIOUSB_FindDevicesByGroup . . . 745
AIOUSB_FindDeviceIndicesByGroup . . . 745
NewWDGConfig . . . 745
DeleteWDGConfig . . . 745
WDG_SetConfig . . . 745
WDG_GetStatus . . . 745
WDG_Pet . . . 745
NewUSBDevice . . . 745
DeleteUSBDevice . . . 745
CopyUSBDevice . . . 745
InitializeUSBDevice . . . 745

AddAllACCESUSBDevices . . . 746
DeleteUSBDevices . . . 746
USBDeviceClose . . . 746
USBDeviceGetIdProduct . . . 746
USBDeviceFetchADCConfigBlock . . . 746
USBDevicePutADCConfigBlock . . . 746
usb_control_transfer . . . 746
usb_bulk_transfer . . . 746
usb_request . . . 746
usb_reset_device . . . 746
get_usb_device . . . 746
USBDeviceGetUSBDeviceHandle . . . 746
NewDefaultAIOCommandLineOptions . . . 746
NewAIOCommandLineOptionsFromDefaultOptions . . . 746
AIO_CMDLINE_DEFAULT_OPTIONS . . . 746
AIO_CMDLINE_SCRIPTING_OPTIONS . . . 746
AIO_CMDLINE_CLEAR_OPTIONS . . . 746
AIOProcessCmdline . . . 746
AIOProcessCommandLine . . . 746
AIOGetChannelRange . . . 747
AIOPrintUsage . . . 747
DeleteAIOCommandLineOptions . . . 747
AIOCommandLineOptionsListDevices . . . 747
AIOCommandLineOptionsOverrideADCConfigBlock . . . 747
AIO_SCRIPTING_OPTIONS . . . 747
AIO_CMDLINE_OPTIONS . . . 747
34.206.4 Variable Documentation . . . 747
AD_GAIN_CODE_STRINGS . . . 748
aio_channel_range_error . . . 748
AIOUSB_INIT_PATTERN . . . 748
34.207 /b/ndkbuild/jni/USBDevice.c File Reference . . . 748
34.207.1 Function Documentation . . . 748
InitializeUSBDevice . . . 748
NewUSBDevice . . . 748
CopyUSBDevice . . . 748
USBDeviceClose . . . 748
AddAllACCESUSBDevices . . . 748
AddDevice . . . 749
USBDeviceGetIdProduct . . . 749
DeleteUSBDevices . . . 749
DeleteUSBDevice . . . 749
USBDeviceSetDebug . . . 749
USBDeviceGetUSBDeviceHandle . . . 749
get_usb_device . . . 749
USBDeviceFetchADCConfigBlock . . . 749
USBDevicePutADCConfigBlock . . . 749
usb_control_transfer . . . 749
usb_bulk_transfer . . . 749
usb_request . . . 749

usb_reset_device . . . 749
34.208 lib/USBDevice.c File Reference . . . 749
34.208.1 Function Documentation . . . 750
InitializeUSBDevice . . . 750
NewUSBDevice . . . 750
CopyUSBDevice . . . 750
USBDeviceClose . . . 750
AddAllACCESUSBDevices . . . 750
AddDevice . . . 750
USBDeviceGetIdProduct . . . 750
DeleteUSBDevices . . . 750
DeleteUSBDevice . . . 750
USBDeviceSetDebug . . . 750
USBDeviceGetUSBDeviceHandle . . . 750
get_usb_device . . . 750
USBDeviceFetchADCConfigBlock . . . 750
USBDevicePutADCConfigBlock . . . 750
usb_control_transfer . . . 750
usb_bulk_transfer . . . 750
usb_request . . . 750
usb_reset_device . . . 750
34.209 lib/ndkbuild/jni/USBDevice.h File Reference . . . 750
34.209.1 Macro Definition Documentation . . . 751
INTERNAL_METHOD . . . 751
34.209.2 Typedef Documentation . . . 751
USBDevice . . . 751
LIBUSBArgs . . . 751
34.209.3 Function Documentation . . . 751
NewUSBDevice . . . 751
DeleteUSBDevice . . . 751
CopyUSBDevice . . . 751
InitializeUSBDevice . . . 751
AddAllACCESUSBDevices . . . 752
DeleteUSBDevices . . . 752
USBDeviceClose . . . 752
USBDeviceGetIdProduct . . . 752
USBDeviceFetchADCConfigBlock . . . 752
USBDevicePutADCConfigBlock . . . 752
usb_control_transfer . . . 752
usb_bulk_transfer . . . 752
usb_request . . . 752
usb_reset_device . . . 752
get_usb_device . . . 752
USBDeviceGetUSBDeviceHandle . . . 752
34.210 lib/USBDevice.h File Reference . . . 752
34.210.1 Macro Definition Documentation . . . 753
INTERNAL_METHOD . . . 753
34.210.2 Typedef Documentation . . . 753
USBDevice . . . 753

LIBUSBArgs . . . 753
34.210.3 Function Documentation . . . 753
NewUSBDevice . . . 753
DeleteUSBDevice . . . 753
CopyUSBDevice . . . 753
InitializeUSBDevice . . . 753
AddAllACCESUSBDevices . . . 753
DeleteUSBDevices . . . 753
USBDeviceClose . . . 753
USBDeviceGetIdProduct . . . 753
USBDeviceFetchADCConfigBlock . . . 753
USBDevicePutADCConfigBlock . . . 753
usb_control_transfer . . . 753
usb_bulk_transfer . . . 753
usb_request . . . 754
usb_reset_device . . . 754
get_usb_device . . . 754
USBDeviceGetUSBDeviceHandle . . . 754
34.211 lib/ndkbuild/libusb/android/config.h File Reference . . . 754
34.211.1 Macro Definition Documentation . . . 754
ENABLE_LOGGING . . . 754
HAVE_DLFCN_H . . . 754
HAVE_GETTIMEOFDAY . . . 754
HAVE_INTTYPES_H . . . 754
OS_LINUX . . . 754
USE_SYSTEM_LOGGING_FACILITY . . . 754
POLL_NFDS_TYPE . . . 754
THREADS_POSIX . . . 754
DEFAULT_VISIBILITY . . . 754
HAVE_MEMORY_H . . . 754
HAVE_POLL_H . . . 754
HAVE_SIGNAL_H . . . 754
HAVE_SYS_STAT_H . . . 754
HAVE_SYS_TIME_H . . . 754
HAVE_SYS_TYPES_H . . . 754
HAVE_UNISTD_H . . . 754
HAVE_LINUX_FILTER_H . . . 755
HAVE_LINUX_NETLINK_H . . . 755
HAVE_ASM_TYPES_H . . . 755
HAVE_SYS_SOCKET_H . . . 755
34.212 lib/ndkbuild/libusb/msvc/config.h File Reference . . . 755
34.212.1 Macro Definition Documentation . . . 755
DEFAULT_VISIBILITY . . . 755
ENABLE_LOGGING . . . 755
POLL_NFDS_TYPE . . . 755
OS_WINDOWS . . . 755
HAVE_SIGNAL_H . . . 755
HAVE_SYS_TYPES_H . . . 755
34.213 lib/ndkbuild/libusb/Xcode/config.h File Reference . . . 755

34.213.1 Macro Definition Documentation . . . 755
DEFAULT_VISIBILITY . . . 755
ENABLE_LOGGING . . . 755
HAVE_GETTIMEOFDAY . . . 755
HAVE_POLL_H . . . 755
HAVE_SYS_TIME_H . . . 755
OS_DARWIN . . . 755
POLL_NFDS_TYPE . . . 755
THREADS_POSIX . . . 755
34.214 lib/ndkbuild/libusb/examples/dpfp.c File Reference . . . 756
34.214.1 Macro Definition Documentation . . . 756
EP_INTR . . . 756
EP_DATA . . . 756
CTRL_IN . . . 756
CTRL_OUT . . . 756
USB_RQ . . . 756
INTR_LENGTH . . . 756
34.214.2 Enumeration Type Documentation . . . 756
anonymous enum . . . 756
anonymous enum . . . 757
34.214.3 Function Documentation . . . 757
main . . . 757
34.215 lib/ndkbuild/libusb/examples/dpfp_threaded.c File Reference . . . 757
34.215.1 Macro Definition Documentation . . . 757
EP_INTR . . . 757
EP_DATA . . . 757
CTRL_IN . . . 757
CTRL_OUT . . . 757
USB_RQ . . . 757
INTR_LENGTH . . . 758
SEM_NAME . . . 758
34.215.2 Enumeration Type Documentation . . . 758
anonymous enum . . . 758
anonymous enum . . . 758
34.215.3 Function Documentation . . . 758
main . . . 758
34.216 lib/ndkbuild/libusb/examples/ezusb.c File Reference . . . 758
34.216.1 Macro Definition Documentation . . . 759
RW_INTERNAL . . . 759
RW_MEMORY . . . 759
RETRY_LIMIT . . . 759
34.216.2 Enumeration Type Documentation . . . 759
ram_mode . . . 759
34.216.3 Function Documentation . . . 759
logerror . . . 759
ezusb_load_ram . . . 759
34.216.4 Variable Documentation . . . 759
verbose . . . 759
34.217 lib/ndkbuild/libusb/examples/ezusb.h File Reference . . . 759

34.217.1 Macro Definition Documentation . . . 760
FX_TYPE_UNDEFINED . . . 760
FX_TYPE_AN21 . . . 760
FX_TYPE_FX1 . . . 760
FX_TYPE_FX2 . . . 760
FX_TYPE_FX2LP . . . 760
FX_TYPE_FX3 . . . 760
FX_TYPE_MAX . . . 760
FX_TYPE_NAMES . . . 760
IMG_TYPE_UNDEFINED . . . 760
IMG_TYPE_HEX . . . 760
IMG_TYPE_IIC . . . 760
IMG_TYPE_BIX . . . 760
IMG_TYPE_IMG . . . 760
IMG_TYPE_MAX . . . 760
IMG_TYPE_NAMES . . . 760
FX_KNOWN_DEVICES . . . 760
34.217.2 Function Documentation . . . 760
ezusb_load_ram . . . 760
ezusb_load_eeprom . . . 760
34.217.3 Variable Documentation . . . 760
verbose . . . 760
34.218 libusb/ndkbuild/libusb/examples/fxload.c File Reference . . . 761
34.218.1 Macro Definition Documentation . . . 761
FXLOAD_VERSION . . . 761
ARRAYSIZE . . . 761
FIRMWARE . . . 761
LOADER . . . 761
34.218.2 Function Documentation . . . 761
logerror . . . 761
main . . . 761
34.219 libusb/ndkbuild/libusb/examples/getopt/getopt.c File Reference . . . 761
34.219.1 Macro Definition Documentation . . . 762
const . . . 762
GETOPT_INTERFACE_VERSION . . . 762
SWAP_FLAGS . . . 762
NONOPTION_P . . . 762
34.219.2 Enumeration Type Documentation . . . 762
anonymous enum . . . 762
34.219.3 Function Documentation . . . 762
getenv . . . 762
getopt . . . 762
34.219.4 Variable Documentation . . . 762
optarg . . . 762
optind . . . 762
opterr . . . 762
optopt . . . 762
34.220 libusb/ndkbuild/libusb/examples/getopt/getopt.h File Reference . . . 762
34.220.1 Macro Definition Documentation . . . 763

no_argument . . . 763
required_argument . . . 763
optional_argument . . . 763
34.220.2 Function Documentation . . . 763
getopt . . . 763
getopt_long . . . 763
getopt_long_only . . . 763
34.220.3 Variable Documentation . . . 763
optarg . . . 763
optind . . . 763
opterr . . . 763
optopt . . . 763
34.221 lib/ndkbuild/libusb/examples/getopt/getopt1.c File Reference . . . 763
34.221.1 Macro Definition Documentation . . . 763
const . . . 763
GETOPT_INTERFACE_VERSION . . . 763
NULL . . . 763
34.221.2 Function Documentation . . . 763
getopt_long . . . 763
getopt_long_only . . . 763
34.222 lib/ndkbuild/libusb/examples/hotplugtest.c File Reference . . . 764
34.222.1 Function Documentation . . . 764
main . . . 764
34.222.2 Variable Documentation . . . 764
done . . . 764
handle . . . 764
34.223 lib/ndkbuild/libusb/examples/listdevs.c File Reference . . . 764
34.223.1 Function Documentation . . . 764
main . . . 764
34.224 lib/ndkbuild/libusb/examples/sam3u_benchmark.c File Reference . . . 764
34.224.1 Macro Definition Documentation . . . 765
EP_DATA_IN . . . 765
EP_ISO_IN . . . 765
34.224.2 Function Documentation . . . 765
main . . . 765
34.225 lib/ndkbuild/libusb/examples/testlibusb.c File Reference . . . 765
34.225.1 Function Documentation . . . 765
main . . . 765
34.225.2 Variable Documentation . . . 765
verbose . . . 765
34.226 lib/ndkbuild/libusb/examples/xusb.c File Reference . . . 765
34.226.1 Macro Definition Documentation . . . 766
msleep . . . 766
bool . . . 766
true . . . 766
false . . . 766
usb_interface . . . 766
ERR_EXIT . . . 766
CALL_CHECK . . . 766

B . . . 766
be_to_int32 . . . 766
RETRY_MAX . . . 766
REQUEST_SENSE_LENGTH . . . 766
INQUIRY_LENGTH . . . 766
READ_CAPACITY_LENGTH . . . 766
HID_GET_REPORT . . . 766
HID_GET_IDLE . . . 766
HID_GET_PROTOCOL . . . 766
HID_SET_REPORT . . . 766
HID_SET_IDLE . . . 766
HID_SET_PROTOCOL . . . 766
HID_REPORT_TYPE_INPUT . . . 766
HID_REPORT_TYPE_OUTPUT . . . 766
HID_REPORT_TYPE_FEATURE . . . 767
BOMS_RESET . . . 767
BOMS_GET_MAX_LUN . . . 767
MAX_OS_FD_LENGTH . . . 767
34.226.2 Enumeration Type Documentation . . . 767
test_type . . . 767
34.226.3 Function Documentation . . . 767
main . . . 767
34.227 /home/b/ndkbuild/libusb/libusb/core.c File Reference . . . 767
34.227.1 Macro Definition Documentation . . . 769
DISCOVERED_DEVICES_SIZE_STEP . . . 769
34.227.2 Function Documentation . . . 769
discovered_devs_append . . . 769
usbi_alloc_device . . . 769
usbi_connect_device . . . 769
usbi_disconnect_device . . . 769
usbi_sanitize_device . . . 769
usbi_get_device_by_session_id . . . 769
usbi_signal_event . . . 769
usbi_clear_event . . . 769
usbi_log_v . . . 769
usbi_log . . . 769
34.227.3 Variable Documentation . . . 769
usbi_default_context . . . 769
active_contexts_lock . . . 769
active_contexts_list . . . 769
34.228 /home/b/ndkbuild/libusb/libusb/descriptor.c File Reference . . . 769
34.228.1 Macro Definition Documentation . . . 771
DESC_HEADER_LENGTH . . . 771
DEVICE_DESC_LENGTH . . . 771
CONFIG_DESC_LENGTH . . . 771
INTERFACE_DESC_LENGTH . . . 771
ENDPOINT_DESC_LENGTH . . . 771
ENDPOINT_AUDIO_DESC_LENGTH . . . 771
34.228.2 Function Documentation . . . 771

usbi_parse_descriptor . . . 771
usbi_device_cache_descriptor . . . 771
usbi_get_config_index_by_value . . . 771
34.229 lib/ndkbuild/libusb/libusb/hotplug.c File Reference . . . 771
34.229.1 Function Documentation . . . 771
usbi_hotplug_match . . . 771
usbi_hotplug_notification . . . 771
usbi_hotplug_deregister_all . . . 771
34.230 lib/ndkbuild/libusb/libusb/hotplug.h File Reference . . . 771
34.230.1 Typedef Documentation . . . 772
libusb_hotplug_callback . . . 772
libusb_hotplug_message . . . 772
34.230.2 Function Documentation . . . 772
usbi_hotplug_deregister_all . . . 772
usbi_hotplug_match . . . 772
usbi_hotplug_notification . . . 772
34.231 lib/ndkbuild/libusb/libusb/io.c File Reference . . . 772
34.231.1 Function Documentation . . . 773
usbi_io_init . . . 773
usbi_io_exit . . . 773
usbi_handle_transfer_completion . . . 773
usbi_handle_transfer_cancellation . . . 773
usbi_signal_transfer_completion . . . 773
usbi_add_pollfd . . . 773
usbi_remove_pollfd . . . 773
usbi_handle_disconnect . . . 773
34.232 lib/ndkbuild/libusb/libusb/libusb.h File Reference . . . 773
34.232.1 Macro Definition Documentation . . . 780
LIBUSB_DEPRECATED_FOR . . . 780
LIBUSBX_API_VERSION . . . 780
LIBUSB_DT_DEVICE_SIZE . . . 780
LIBUSB_DT_CONFIG_SIZE . . . 780
LIBUSB_DT_INTERFACE_SIZE . . . 780
LIBUSB_DT_ENDPOINT_SIZE . . . 780
LIBUSB_DT_ENDPOINT_AUDIO_SIZE . . . 780
LIBUSB_DT_HUB_NONVAR_SIZE . . . 780
LIBUSB_DT_SS_ENDPOINT_COMPANION_SIZE . . . 780
LIBUSB_DT_BOS_SIZE . . . 780
LIBUSB_DT_DEVICE_CAPABILITY_SIZE . . . 780
LIBUSB_BT_USB_2_0_EXTENSION_SIZE . . . 780
LIBUSB_BT_SS_USB_DEVICE_CAPABILITY_SIZE . . . 780
LIBUSB_BT_CONTAINER_ID_SIZE . . . 780
LIBUSB_DT_BOS_MAX_SIZE . . . 780
LIBUSB_ENDPOINT_ADDRESS_MASK . . . 780
LIBUSB_ENDPOINT_DIR_MASK . . . 780
LIBUSB_TRANSFER_TYPE_MASK . . . 780
LIBUSB_ISO_SYNC_TYPE_MASK . . . 780
LIBUSB_ISO_USAGE_TYPE_MASK . . . 780
LIBUSB_CONTROL_SETUP_SIZE . . . 780

LIBUSB_ERROR_COUNT . . . 780
34.233 libusb/ndkbuild/libusb/libusb/libusbi.h File Reference . . . 781
34.233.1 Macro Definition Documentation . . . 782
API_EXPORTED . . . 782
DEVICE_DESC_LENGTH . . . 782
USB_MAXENDPOINTS . . . 782
USB_MAXINTERFACES . . . 782
USB_MAXCONFIG . . . 782
USBI_CAP_HAS_HID_ACCESS . . . 782
USBI_CAP_SUPPORTS_DETACH_KERNEL_DRIVER . . . 782
USBI_MAX_LOG_LEN . . . 783
USBI_LOG_LINE_END . . . 783
UNUSED . . . 783
ARRAYSIZE . . . 783
list_entry . . . 783
list_first_entry . . . 783
list_for_each_entry . . . 783
list_for_each_entry_safe . . . 783
list_empty . . . 783
container_of . . . 783
MIN . . . 783
MAX . . . 783
TIMESPEC_IS_SET . . . 783
TIMEVAL_TV_SEC_TYPE . . . 783
TIMESPEC_TO_TIMEVAL . . . 783
usbi_dbg . . . 783
usbi_info . . . 783
usbi_warn . . . 783
usbi_err . . . 783
USBI_GET_CONTEXT . . . 783
DEVICE_CTX . . . 784
HANDLE_CTX . . . 784
TRANSFER_CTX . . . 784
ITRANSFER_CTX . . . 784
IS_EPIN . . . 784
IS_EPOUT . . . 784
IS_XFERIN . . . 784
IS_XFEROUT . . . 784
usbi_handling_events . . . 784
usbi_start_event_handling . . . 784
usbi_end_event_handling . . . 784
usbi_pending_events . . . 784
usbi_using_timerfd . . . 784
USBI_TRANSFER_TO_LIBUSB_TRANSFER . . . 784
LIBUSB_TRANSFER_TO_USBI_TRANSFER . . . 784
34.233.2 Enumeration Type Documentation . . . 784
usbi_event_flags . . . 784
anonymous enum . . . 784
usbi_transfer_state_flags . . . 784

usbi_transfer_timeout_flags . . . 785
34.233.3 Function Documentation . . . 785
usbi_log . . . 785
usbi_log_v . . . 785
usbi_io_init . . . 785
usbi_io_exit . . . 785
usbi_alloc_device . . . 785
usbi_get_device_by_session_id . . . 785
usbi_sanitize_device . . . 785
usbi_handle_disconnect . . . 785
usbi_handle_transfer_completion . . . 785
usbi_handle_transfer_cancellation . . . 785
usbi_signal_transfer_completion . . . 785
usbi_parse_descriptor . . . 785
usbi_device_cache_descriptor . . . 785
usbi_get_config_index_by_value . . . 785
usbi_connect_device . . . 785
usbi_disconnect_device . . . 785
usbi_signal_event . . . 785
usbi_clear_event . . . 785
usbi_add_pollfd . . . 785
usbi_remove_pollfd . . . 785
discovered_devs_append . . . 785
34.233.4 Variable Documentation . . . 785
usbi_default_context . . . 785
usbi_backend . . . 785
linux_usbfs_backend . . . 785
darwin_backend . . . 785
openbsd_backend . . . 785
netbsd_backend . . . 785
windows_backend . . . 785
usbdk_backend . . . 786
wince_backend . . . 786
haiku_usb_raw_backend . . . 786
sunos_backend . . . 786
active_contexts_list . . . 786
active_contexts_lock . . . 786
34.234 lib/ndkbuild/libusb/libusb/os/darwin_usb.c File Reference . . . 786
34.234.1 Macro Definition Documentation . . . 786
DARWIN_CACHED_DEVICE . . . 786
34.234.2 Variable Documentation . . . 786
darwin_backend . . . 786
34.235 lib/ndkbuild/libusb/libusb/os/darwin_usb.h File Reference . . . 786
34.235.1 Macro Definition Documentation . . . 787
IO_OBJECT_NULL . . . 787
34.235.2 Typedef Documentation . . . 787
io_cf_plugin_ref_t . . . 787
io_notification_port_t . . . 787
34.236 lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp File Reference . . . 787

34.237 lib/ndkbuild/libusb/libusb/os/haiku_usb.h File Reference . . . 787
34.238 lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp File Reference . . . 788
34.239 lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.cpp File Reference . . . 788
34.239.1 Variable Documentation . . . 788
gUsbRoster . . . 788
gInitCount . . . 788
haiku_usb_raw_backend . . . 788
34.240 lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.h File Reference . . . 788
34.240.1 Macro Definition Documentation . . . 789
B_USB_RAW_PROTOCOL_VERSION . . . 789
B_USB_RAW_ACTIVE_ALTERNATE . . . 789
34.240.2 Enumeration Type Documentation . . . 789
usb_raw_command_id . . . 789
usb_raw_command_status . . . 789
34.241 lib/ndkbuild/libusb/libusb/os/linux_netlink.c File Reference . . . 790
34.241.1 Macro Definition Documentation . . . 790
NL_GROUP_KERNEL . . . 790
34.241.2 Function Documentation . . . 790
linux_netlink_start_event_monitor . . . 790
linux_netlink_stop_event_monitor . . . 790
linux_netlink_hotplug_poll . . . 790
34.242 lib/ndkbuild/libusb/libusb/os/linux_udev.c File Reference . . . 790
34.242.1 Function Documentation . . . 791
linux_udev_start_event_monitor . . . 791
linux_udev_stop_event_monitor . . . 791
linux_udev_scan_devices . . . 791
linux_udev_hotplug_poll . . . 791
34.243 lib/ndkbuild/libusb/libusb/os/linux_usbfs.c File Reference . . . 791
34.243.1 Enumeration Type Documentation . . . 791
reap_action . . . 791
34.243.2 Function Documentation . . . 792
linux_get_device_address . . . 792
linux_enumerate_device . . . 792
linux_hotplug_enumerate . . . 792
linux_device_disconnected . . . 792
34.243.3 Variable Documentation . . . 792
linux_hotplug_lock . . . 792
linux_usbfs_backend . . . 792
34.244 lib/ndkbuild/libusb/libusb/os/linux_usbfs.h File Reference . . . 792
34.244.1 Macro Definition Documentation . . . 793
SYSFS_DEVICE_PATH . . . 793
USBFS_MAXDRIVERNAME . . . 793
USBFS_URB_SHORT_NOT_OK . . . 793
USBFS_URB_ISO_ASAP . . . 793
USBFS_URB_BULK_CONTINUATION . . . 793
USBFS_URB_QUEUE_BULK . . . 793
USBFS_URB_ZERO_PACKET . . . 793
MAX_ISO_BUFFER_LENGTH . . . 793
MAX_BULK_BUFFER_LENGTH . . . 793

MAX_CTRL_BUFFER_LENGTH . . . 793
USBFS_CAP_ZERO_PACKET . . . 793
USBFS_CAP_BULK_CONTINUATION . . . 793
USBFS_CAP_NO_PACKET_SIZE_LIM . . . 794
USBFS_CAP_BULK_SCATTER_GATHER . . . 794
USBFS_CAP_REAP_AFTER_DISCONNECT . . . 794
USBFS_DISCONNECT_CLAIM_IF_DRIVER . . . 794
USBFS_DISCONNECT_CLAIM_EXCEPT_DRIVER . . . 794
IOCTL_USBFS_CONTROL . . . 794
IOCTL_USBFS_BULK . . . 794
IOCTL_USBFS_RESETEP . . . 794
IOCTL_USBFS_SETINTF . . . 794
IOCTL_USBFS_SETCONFIG . . . 794
IOCTL_USBFS_GETDRIVER . . . 794
IOCTL_USBFS_SUBMITURB . . . 794
IOCTL_USBFS_DISCARDURB . . . 794
IOCTL_USBFS_REAPURB . . . 794
IOCTL_USBFS_REAPURBNDELAY . . . 794
IOCTL_USBFS_CLAIMINTF . . . 794
IOCTL_USBFS_RELEASEINTF . . . 794
IOCTL_USBFS_CONNECTINFO . . . 794
IOCTL_USBFS_IOCTL . . . 794
IOCTL_USBFS_HUB_PORTINFO . . . 794
IOCTL_USBFS_RESET . . . 794
IOCTL_USBFS_CLEAR_HALT . . . 794
IOCTL_USBFS_DISCONNECT . . . 794
IOCTL_USBFS_CONNECT . . . 794
IOCTL_USBFS_CLAIM_PORT . . . 794
IOCTL_USBFS_RELEASE_PORT . . . 794
IOCTL_USBFS_GET_CAPABILITIES . . . 794
IOCTL_USBFS_DISCONNECT_CLAIM . . . 794
IOCTL_USBFS_ALLOC_STREAMS . . . 794
IOCTL_USBFS_FREE_STREAMS . . . 794
34.244.2 Enumeration Type Documentation . . . 794
usbfs_urb_type . . . 794
34.244.3 Function Documentation . . . 795
linux_netlink_start_event_monitor . . . 795
linux_netlink_stop_event_monitor . . . 795
linux_netlink_hotplug_poll . . . 795
linux_hotplug_enumerate . . . 795
linux_device_disconnected . . . 795
linux_get_device_address . . . 795
linux_enumerate_device . . . 795
34.244.4 Variable Documentation . . . 795
linux_hotplug_lock . . . 795
34.245 libusb/ndkbuild/libusb/libusb/os/netbsd_usb.c File Reference . . . 795
34.245.1 Variable Documentation . . . 795
netbsd_backend . . . 795
34.246 libusb/ndkbuild/libusb/libusb/os/openbsd_usb.c File Reference . . . 795

34.246.1 Macro Definition Documentation . . . 796
DEVPATH . . . 796
USBDEV . . . 796
34.246.2 Variable Documentation . . . 796
openbsd_backend . . . 796
34.247 lib/ndkbuild/libusb/libusb/os/poll_posix.c File Reference . . . 796
34.247.1 Function Documentation . . . 796
usbi_pipe . . . 796
34.248 lib/ndkbuild/libusb/libusb/os/poll_posix.h File Reference . . . 796
34.248.1 Macro Definition Documentation . . . 796
usbi_write . . . 796
usbi_read . . . 796
usbi_close . . . 796
usbi_poll . . . 797
34.248.2 Function Documentation . . . 797
usbi_pipe . . . 797
34.249 lib/ndkbuild/libusb/libusb/os/poll_windows.c File Reference . . . 797
34.249.1 Macro Definition Documentation . . . 797
poll_dbg . . . 797
CHECK_INIT_POLLING . . . 797
Use_Duplicate_Handles . . . 797
34.249.2 Function Documentation . . . 797
init_polling . . . 797
exit_polling . . . 797
usbi_pipe . . . 797
usbi_create_fd . . . 797
usbi_free_fd . . . 797
fd_to_winfd . . . 798
handle_to_winfd . . . 798
overlapped_to_winfd . . . 798
usbi_poll . . . 798
usbi_close . . . 798
usbi_write . . . 798
usbi_read . . . 798
34.249.3 Variable Documentation . . . 798
INVALID_WINFD . . . 798
poll_fd . . . 798
mutex . . . 798
original_handle . . . 798
thread_id . . . 798
is_polling_set . . . 798
pipe_number . . . 798
LPOVERLAPPED . . . 798
34.250 lib/ndkbuild/libusb/libusb/os/poll_windows.h File Reference . . . 798
34.250.1 Macro Definition Documentation . . . 799
STATUS_REPARSE . . . 799
STATUS_COMPLETED_SYNCHRONOUSLY . . . 799
HasOverlappedIoCompletedSync . . . 799
DUMMY_HANDLE . . . 799

MAX_FDS . . . 799
POLLIN . . . 799
POLLPRI . . . 799
POLLOUT . . . 799
POLLERR . . . 799
POLLHUP . . . 799
POLLNVAL . . . 799
TIMESPEC_TO_TIMEVAL . . . 799
timersub . . . 799
34.250.2 Typedef Documentation . . . 800
cancel_transfer . . . 800
34.250.3 Enumeration Type Documentation . . . 800
windows_version . . . 800
rw_type . . . 800
34.250.4 Function Documentation . . . 800
usbi_pipe . . . 800
usbi_poll . . . 800
usbi_write . . . 800
usbi_read . . . 800
usbi_close . . . 800
init_polling . . . 800
exit_polling . . . 800
usbi_create_fd . . . 800
usbi_free_fd . . . 800
fd_to_winfd . . . 800
handle_to_winfd . . . 800
overlapped_to_winfd . . . 800
34.250.5 Variable Documentation . . . 800
windows_version . . . 800
INVALID_WINFD . . . 800
34.251 lib/ndkbuild/libusb/libusb/os/sunos_usb.c File Reference . . . 801
34.251.1 Variable Documentation . . . 801
sunos_backend . . . 801
34.252 lib/ndkbuild/libusb/libusb/os/sunos_usb.h File Reference . . . 801
34.252.1 Macro Definition Documentation . . . 802
READ . . . 802
WRITE . . . 802
34.252.2 Typedef Documentation . . . 802
sunos_dev_priv_t . . . 802
sunos_ep_priv_t . . . 802
sunos_dev_handle_priv_t . . . 802
sunos_xfer_priv_t . . . 802
34.253 lib/ndkbuild/libusb/libusb/os/threads_posix.c File Reference . . . 802
34.253.1 Function Documentation . . . 802
usbi_cond_timedwait . . . 802
usbi_get_tid . . . 802
34.254 lib/ndkbuild/libusb/libusb/os/threads_posix.h File Reference . . . 802
34.254.1 Macro Definition Documentation . . . 803
usbi_mutex_static_t . . . 803

USBI_MUTEX_INITIALIZER . . . 803
usbi_mutex_static_lock . . . 803
usbi_mutex_static_unlock . . . 803
usbi_mutex_t . . . 803
usbi_mutex_init . . . 803
usbi_mutex_lock . . . 803
usbi_mutex_unlock . . . 803
usbi_mutex_trylock . . . 803
usbi_mutex_destroy . . . 803
usbi_cond_t . . . 803
usbi_cond_init . . . 803
usbi_cond_wait . . . 803
usbi_cond_broadcast . . . 803
usbi_cond_destroy . . . 803
usbi_tls_key_t . . . 803
usbi_tls_key_create . . . 803
usbi_tls_key_get . . . 803
usbi_tls_key_set . . . 803
usbi_tls_key_delete . . . 803
34.254.2 Function Documentation . . . 803
usbi_cond_timedwait . . . 803
usbi_get_tid . . . 803
34.255 libusb/ndkbuild/libusb/os/threads_windows.c File Reference . . . 803
34.255.1 Function Documentation . . . 804
usbi_mutex_static_lock . . . 804
usbi_mutex_static_unlock . . . 804
usbi_mutex_init . . . 804
usbi_mutex_lock . . . 804
usbi_mutex_unlock . . . 804
usbi_mutex_trylock . . . 804
usbi_mutex_destroy . . . 804
usbi_cond_init . . . 804
usbi_cond_destroy . . . 804
usbi_cond_broadcast . . . 804
usbi_cond_wait . . . 804
usbi_cond_timedwait . . . 804
usbi_tls_key_create . . . 804
usbi_tls_key_get . . . 804
usbi_tls_key_set . . . 804
usbi_tls_key_delete . . . 804
usbi_get_tid . . . 804
34.256 libusb/ndkbuild/libusb/os/threads_windows.h File Reference . . . 804
34.256.1 Macro Definition Documentation . . . 805
usbi_mutex_static_t . . . 805
USBI_MUTEX_INITIALIZER . . . 805
usbi_mutex_t . . . 805
HAVE_STRUCT_TIMESPEC . . . 805
ETIMEDOUT . . . 805
usbi_tls_key_t . . . 805

34.256.2 Typedef Documentation . . . 805
usbi_cond_t . . . 805
34.256.3 Function Documentation . . . 805
usbi_mutex_static_lock . . . 805
usbi_mutex_static_unlock . . . 805
usbi_mutex_init . . . 805
usbi_mutex_lock . . . 805
usbi_mutex_unlock . . . 805
usbi_mutex_trylock . . . 805
usbi_mutex_destroy . . . 805
usbi_cond_init . . . 806
usbi_cond_wait . . . 806
usbi_cond_timedwait . . . 806
usbi_cond_broadcast . . . 806
usbi_cond_destroy . . . 806
usbi_tls_key_create . . . 806
usbi_tls_key_get . . . 806
usbi_tls_key_set . . . 806
usbi_tls_key_delete . . . 806
usbi_get_tid . . . 806
34.257 libusb/ndkbuild/libusb/libusb/os/wince_usb.c File Reference . . . 806
34.257.1 Variable Documentation . . . 806
epoch_time . . . 806
windows_version . . . 806
wince_backend . . . 806
34.258 libusb/ndkbuild/libusb/libusb/os/wince_usb.h File Reference . . . 806
34.258.1 Macro Definition Documentation . . . 807
MAX_DEVICE_COUNT . . . 807
UKW_TF_IN_TRANSFER . . . 807
UKW_TF_OUT_TRANSFER . . . 807
UKW_TF_NO_WAIT . . . 807
UKW_TF_SHORT_TRANSFER_OK . . . 807
UKW_TF_SEND_TO_DEVICE . . . 807
UKW_TF_SEND_TO_INTERFACE . . . 807
UKW_TF_SEND_TO_ENDPOINT . . . 808
UKW_TF_DONT_BLOCK_FOR_MEM . . . 808
UKW_ACTIVE_CONFIGURATION . . . 808
STATUS_HALT_FLAG . . . 808
34.258.2 Typedef Documentation . . . 808
UKW_DEVICE . . . 808
PUKW_DEVICE . . . 808
LPUKW_DEVICE . . . 808
PUKW_DEVICE_DESCRIPTOR . . . 808
LPUKW_DEVICE_DESCRIPTOR . . . 808
PUKW_CONTROL_HEADER . . . 808
LPUKW_CONTROL_HEADER . . . 808
34.258.3 Function Documentation . . . 808
DLL_DECLARE_HANDLE . . . 808
DLL_DECLARE_FUNC . . . 808

DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 808
DLL_DECLARE_FUNC . . . 809
DLL_DECLARE_FUNC . . . 809
34.259 libusb/ndkbuild/libusb/libusb/os/windows_common.h File Reference . . . 809
34.259.1 Macro Definition Documentation . . . 809
bool . . . 809
true . . . 809
false . . . 809
safe_free . . . 809
safe_closehandle . . . 809
safe_min . . . 809
safe_strcp . . . 809
safe_strcpy . . . 810
safe_strncat . . . 810
safe_strcat . . . 810
safe_strcmp . . . 810
safe_stricmp . . . 810
safe_strncmp . . . 810
safe_strlen . . . 810
safe_sprintf . . . 810
safe_stprintf . . . 810
safe_tcslen . . . 810
safe_unref_device . . . 810
wchar_to_utf8_ms . . . 810
ARRAYSIZE . . . 810
ERR_BUFFER_SIZE . . . 810
DLL_STRINGIFY . . . 810
DLL_LOAD_LIBRARY . . . 810
DLL_DECLARE_HANDLE . . . 810
DLL_GET_HANDLE . . . 810
DLL_FREE_HANDLE . . . 810

DLL_DECLARE_FUNC_PREFIXNAME . . . 810
DLL_DECLARE_FUNC . . . 810
DLL_DECLARE_FUNC_PREFIXED . . . 810
DLL_LOAD_FUNC_PREFIXNAME . . . 810
DLL_LOAD_FUNC . . . 811
DLL_LOAD_FUNC_PREFIXED . . . 811
34.260 lib/ndkbuild/libusb/libusb/os/windows_nt_common.c File Reference . . . 811
34.260.1 Macro Definition Documentation . . . 812
TIMER_REQUEST_RETRY_MS . . . 812
WM_TIMER_REQUEST . . . 812
WM_TIMER_EXIT . . . 812
HTAB_SIZE . . . 812
34.260.2 Typedef Documentation . . . 812
htab_entry . . . 812
34.260.3 Function Documentation . . . 812
DLL_DECLARE_HANDLE . . . 812
DLL_DECLARE_FUNC_PREFIXED . . . 812
DLL_DECLARE_FUNC_PREFIXED . . . 812
DLL_DECLARE_FUNC_PREFIXED . . . 812
htab_hash . . . 812
windows_destroy_clock . . . 812
windows_clock_gettime . . . 812
windows_handle_callback . . . 812
windows_handle_events . . . 812
windows_common_init . . . 812
windows_common_exit . . . 812
34.260.4 Variable Documentation . . . 812
epoch_time . . . 812
34.261 lib/ndkbuild/libusb/libusb/os/windows_nt_common.h File Reference . . . 812
34.261.1 Macro Definition Documentation . . . 813
FACILITY_SETUPAPI . . . 813
34.261.2 Typedef Documentation . . . 813
USB_CONFIGURATION_DESCRIPTOR . . . 813
PUSB_CONFIGURATION_DESCRIPTOR . . . 813
USB_DEVICE_DESCRIPTOR . . . 813
PUSB_DEVICE_DESCRIPTOR . . . 813
34.261.3 Function Documentation . . . 813
windows_common_init . . . 813
windows_common_exit . . . 813
htab_hash . . . 813
windows_clock_gettime . . . 813
windows_clear_transfer_priv . . . 813
windows_copy_transfer_data . . . 813
windows_get_fd . . . 813
windows_get_overlapped_result . . . 813
windows_handle_callback . . . 813
windows_handle_events . . . 813
34.262 lib/ndkbuild/libusb/libusb/os/windows_usbdk.c File Reference . . . 813
34.263 lib/ndkbuild/libusb/libusb/os/windows_usbdk.h File Reference . . . 813

34.263.1 Typedef Documentation . . . 814
USB_DK_DEVICE_ID . . . 814
PUSB_DK_DEVICE_ID . . . 814
USB_DK_DEVICE_INFO . . . 814
PUSB_DK_DEVICE_INFO . . . 814
USB_DK_CONFIG_DESCRIPTOR_REQUEST . . . 815
PUSB_DK_CONFIG_DESCRIPTOR_REQUEST . . . 815
USB_DK_ISO_TRANSFER_RESULT . . . 815
PUSB_DK_ISO_TRANSFER_RESULT . . . 815
USB_DK_GEN_TRANSFER_RESULT . . . 815
PUSB_DK_GEN_TRANSFER_RESULT . . . 815
USB_DK_TRANSFER_RESULT . . . 815
PUSB_DK_TRANSFER_RESULT . . . 815
USB_DK_TRANSFER_REQUEST . . . 815
PUSB_DK_TRANSFER_REQUEST . . . 815
USBDK_GET_DEVICES_LIST . . . 815
USBDK_RELEASE_DEVICES_LIST . . . 815
USBDK_START_REDIRECT . . . 815
USBDK_STOP_REDIRECT . . . 815
USBDK_GET_CONFIGURATION_DESCRIPTOR . . . 815
USBDK_RELEASE_CONFIGURATION_DESCRIPTOR . . . 815
USBDK_WRITE_PIPE . . . 815
USBDK_READ_PIPE . . . 815
USBDK_ABORT_PIPE . . . 815
USBDK_RESET_PIPE . . . 815
USBDK_SET_ALTSETTING . . . 815
USBDK_RESET_DEVICE . . . 815
USBDK_GET_REDIRECTOR_SYSTEM_HANDLE . . . 815
34.263.2 Enumeration Type Documentation . . . 815
TransferResult . . . 815
USB_DK_DEVICE_SPEED . . . 815
USB_DK_TRANSFER_TYPE . . . 816
34.264 lib/ndkbuild/libusb/libusb/os/windows_winusb.c File Reference . . . 816
34.264.1 Macro Definition Documentation . . . 816
HANDLE_VALID . . . 816
LOOP_CHECK . . . 816
LOOP_BREAK . . . 817
CHECK_WINUSBX_AVAILABLE . . . 817
CHECK_HID_AVAILABLE . . . 817
MAX_ENUM_GUIDS . . . 817
HCD_PASS . . . 817
HUB_PASS . . . 817
GEN_PASS . . . 817
DEV_PASS . . . 817
HID_PASS . . . 817
WinUSBX_Set . . . 817
34.264.2 Function Documentation . . . 817
windows_clear_transfer_priv . . . 817
windows_copy_transfer_data . . . 817

windows_get_fd . . . 817
windows_get_overlapped_result . . . 817
34.264.3 Variable Documentation . . . 817
windows_version . . . 818
windows_backend . . . 818
usb_api_backend . . . 818
34.265 usb/ndkbuild/libusb/libusb/os/windows_winusb.h File Reference . . . 818
34.265.1 Macro Definition Documentation . . . 823
SPDRP_ADDRESS . . . 823
SPDRP_INSTALL_STATE . . . 823
MAX_CTRL_BUFFER_LENGTH . . . 823
MAX_USB_DEVICES . . . 823
MAX_USB_STRING_LENGTH . . . 823
MAX_HID_REPORT_SIZE . . . 823
MAX_HID_DESCRIPTOR_SIZE . . . 823
MAX_GUID_STRING_LENGTH . . . 823
MAX_PATH_LENGTH . . . 823
MAX_KEY_LENGTH . . . 823
LIST_SEPARATOR . . . 823
INTERFACE_CLAIMED . . . 824
LIBUSB_COMPLETED . . . 824
USB_API_UNSUPPORTED . . . 824
USB_API_HUB . . . 824
USB_API_COMPOSITE . . . 824
USB_API_WINUSBX . . . 824
USB_API_HID . . . 824
USB_API_MAX . . . 824
USB_API_SET . . . 824
SUB_API_NOTSET . . . 824
SUB_API_LIBUSBK . . . 824
SUB_API_LIBUSB0 . . . 824
SUB_API_WINUSB . . . 824
SUB_API_MAX . . . 824
WINUSBX_DRV_NAMES . . . 824
PRINT_UNSUPPORTED_API . . . 824
LIBUSB_DT_HID_SIZE . . . 824
HID_MAX_CONFIG_DESC_SIZE . . . 824
HID_MAX_REPORT_SIZE . . . 824
HID_IN_EP . . . 824
HID_OUT_EP . . . 824
LIBUSB_REQ_RECIPIENT . . . 824
LIBUSB_REQ_TYPE . . . 824
LIBUSB_REQ_IN . . . 824
LIBUSB_REQ_OUT . . . 824
HID_CTL_CODE . . . 824
HID_BUFFER_CTL_CODE . . . 824
HID_IN_CTL_CODE . . . 824
HID_OUT_CTL_CODE . . . 825
IOCTL_HID_GET_FEATURE . . . 825

IOCTL_HID_GET_INPUT_REPORT . . . 825
IOCTL_HID_SET_FEATURE . . . 825
IOCTL_HID_SET_OUTPUT_REPORT . . . 825
CR_SUCCESS . . . 825
CR_NO_SUCH_DEVNODE . . . 825
USB_DEVICE_DESCRIPTOR_TYPE . . . 825
USB_CONFIGURATION_DESCRIPTOR_TYPE . . . 825
USB_STRING_DESCRIPTOR_TYPE . . . 825
USB_INTERFACE_DESCRIPTOR_TYPE . . . 825
USB_ENDPOINT_DESCRIPTOR_TYPE . . . 825
USB_REQUEST_GET_STATUS . . . 825
USB_REQUEST_CLEAR_FEATURE . . . 825
USB_REQUEST_SET_FEATURE . . . 825
USB_REQUEST_SET_ADDRESS . . . 825
USB_REQUEST_GET_DESCRIPTOR . . . 825
USB_REQUEST_SET_DESCRIPTOR . . . 825
USB_REQUEST_GET_CONFIGURATION . . . 825
USB_REQUEST_SET_CONFIGURATION . . . 825
USB_REQUEST_GET_INTERFACE . . . 825
USB_REQUEST_SET_INTERFACE . . . 825
USB_REQUEST_SYNC_FRAME . . . 825
USB_GET_NODE_INFORMATION . . . 825
USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION . . . 825
USB_GET_NODE_CONNECTION_NAME . . . 825
USB_GET_HUB_CAPABILITIES . . . 825
USB_GET_NODE_CONNECTION_INFORMATION_EX . . . 825
USB_GET_HUB_CAPABILITIES_EX . . . 825
USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 . . . 825
METHOD_BUFFERED . . . 825
FILE_ANY_ACCESS . . . 825
FILE_DEVICE_UNKNOWN . . . 825
FILE_DEVICE_USB . . . 825
CTL_CODE . . . 826
IOCTL_USB_GET_HUB_CAPABILITIES_EX . . . 826
IOCTL_USB_GET_HUB_CAPABILITIES . . . 826
IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION . . . 826
IOCTL_USB_GET_ROOT_HUB_NAME . . . 826
IOCTL_USB_GET_NODE_INFORMATION . . . 826
IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX . . . 826
IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 . . . 826
IOCTL_USB_GET_NODE_CONNECTION_ATTRIBUTES . . . 826
IOCTL_USB_GET_NODE_CONNECTION_NAME . . . 826
SHORT_PACKET_TERMINATE . . . 826
AUTO_CLEAR_STALL . . . 826
PIPE_TRANSFER_TIMEOUT . . . 826
IGNORE_SHORT_PACKETS . . . 826
ALLOW_PARTIAL_READS . . . 826
AUTO_FLUSH . . . 826
RAW_IO . . . 826

MAXIMUM_TRANSFER_SIZE . . . 826
AUTO_SUSPEND . . . 826
SUSPEND_DELAY . . . 826
DEVICE_SPEED . . . 826
LowSpeed . . . 826
FullSpeed . . . 826
HighSpeed . . . 826
HIDP_STATUS_SUCCESS . . . 826
34.265.2 Typedef Documentation . . . 826
DEVNODE . . . 826
DEVINST . . . 826
PDEVNODE . . . 826
PDEVINST . . . 826
RETURN_TYPE . . . 827
CONFIGRET . . . 827
USB_CONNECTION_STATUS . . . 827
PUSB_CONNECTION_STATUS . . . 827
USB_HUB_NODE . . . 827
USB_INTERFACE_DESCRIPTOR . . . 827
PUSB_INTERFACE_DESCRIPTOR . . . 827
USB_CONFIGURATION_DESCRIPTOR_SHORT . . . 827
USB_ENDPOINT_DESCRIPTOR . . . 827
PUSB_ENDPOINT_DESCRIPTOR . . . 827
USB_DESCRIPTOR_REQUEST . . . 827
PUSB_DESCRIPTOR_REQUEST . . . 827
USB_HUB_DESCRIPTOR . . . 827
PUSB_HUB_DESCRIPTOR . . . 827
USB_ROOT_HUB_NAME . . . 827
PUSB_ROOT_HUB_NAME . . . 827
USB_ROOT_HUB_NAME_FIXED . . . 827
USB_NODE_CONNECTION_NAME . . . 827
PUSB_NODE_CONNECTION_NAME . . . 827
USB_NODE_CONNECTION_NAME_FIXED . . . 827
USB_HUB_NAME_FIXED . . . 827
USB_HUB_INFORMATION . . . 827
PUSB_HUB_INFORMATION . . . 827
USB_MI_PARENT_INFORMATION . . . 827
PUSB_MI_PARENT_INFORMATION . . . 827
USB_NODE_INFORMATION . . . 827
PUSB_NODE_INFORMATION . . . 827
USB_PIPE_INFO . . . 827
PUSB_PIPE_INFO . . . 827
USB_NODE_CONNECTION_INFORMATION_EX . . . 827
PUSB_NODE_CONNECTION_INFORMATION_EX . . . 827
USB_PROTOCOLS . . . 827
PUSB_PROTOCOLS . . . 827
USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS . . . 827
PUSB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS . . . 828
USB_NODE_CONNECTION_INFORMATION_EX_V2 . . . 828

PUSB_NODE_CONNECTION_INFORMATION_EX_V2 . . . 828
USB_HUB_CAP_FLAGS . . . 828
PUSB_HUB_CAP_FLAGS . . . 828
USB_HUB_CAPABILITIES . . . 828
PUSB_HUB_CAPABILITIES . . . 828
USB_HUB_CAPABILITIES_EX . . . 828
PUSB_HUB_CAPABILITIES_EX . . . 828
USBD_PIPE_TYPE . . . 828
PWINUSB_PIPE_INFORMATION . . . 828
PWINUSB_SETUP_PACKET . . . 828
WINUSB_INTERFACE_HANDLE . . . 828
PWINUSB_INTERFACE_HANDLE . . . 828
WinUsb_AbortPipe_t . . . 828
WinUsb_ControlTransfer_t . . . 828
WinUsb_FlushPipe_t . . . 828
WinUsb_Free_t . . . 828
WinUsb_GetAssociatedInterface_t . . . 828
WinUsb_GetCurrentAlternateSetting_t . . . 828
WinUsb_GetDescriptor_t . . . 828
WinUsb_GetOverlappedResult_t . . . 828
WinUsb_GetPipePolicy_t . . . 828
WinUsb_GetPowerPolicy_t . . . 828
WinUsb_Initialize_t . . . 828
WinUsb_QueryDeviceInformation_t . . . 828
WinUsb_QueryInterfaceSettings_t . . . 828
WinUsb_QueryPipe_t . . . 828
WinUsb_ReadPipe_t . . . 828
WinUsb_ResetPipe_t . . . 829
WinUsb_SetCurrentAlternateSetting_t . . . 829
WinUsb_SetPipePolicy_t . . . 829
WinUsb_SetPowerPolicy_t . . . 829
WinUsb_WritePipe_t . . . 829
WinUsb_ResetDevice_t . . . 829
KUSB_FNID . . . 829
KLIB_VERSION . . . 829
PKLIB_VERSION . . . 829
LibK_GetProcAddress_t . . . 829
LibK_GetVersion_t . . . 829
PHIDP_PREPARSED_DATA . . . 829
PHIDD_ATTRIBUTES . . . 829
USAGE . . . 829
PHIDP_CAPS . . . 829
HIDP_REPORT_TYPE . . . 829
HIDP_VALUE_CAPS . . . 829
PHIDP_VALUE_CAPS . . . 829
34.265.3 Enumeration Type Documentation . . . 829
libusb_hid_request_type . . . 829
libusb_hid_report_type . . . 829
USB_CONNECTION_STATUS . . . 829

USB_HUB_NODE . . . 830
USBD_PIPE_TYPE . . . 830
34.265.4 Function Documentation . . . 830
DLL_DECLARE_HANDLE . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_HANDLE . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_HANDLE . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_HANDLE . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_FUNC_PREFIXED . . . 830
DLL_DECLARE_HANDLE . . . 830
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_HANDLE . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
DLL_DECLARE_FUNC . . . 831
34.265.5 Variable Documentation . . . 831
GUID_DEVINTERFACE_USB_HOST_CONTROLLER . . . 831
GUID_DEVINTERFACE_USB_DEVICE . . . 831
GUID_DEVINTERFACE_USB_HUB . . . 831
GUID_DEVINTERFACE_LIBUSB0_FILTER . . . 831
usb_api_backend . . . 831
34.266 libusb/ndkbuild/libusb/libusb/strerror.c File Reference . . . 831
34.267 libusb/ndkbuild/libusb/libusb/sync.c File Reference . . . 832

34.268 libusb/ndkbuild/libusb/libusb/version.h File Reference . . . 832
34.268.1 Macro Definition Documentation . . . 832
LIBUSB_MINOR . . . 832
LIBUSB_MICRO . . . 832
LIBUSB_RC . . . 832
34.269 libusb/ndkbuild/libusb/libusb/version_nano.h File Reference . . . 832
34.269.1 Macro Definition Documentation . . . 832
LIBUSB_NANO . . . 832
34.270 libusb/ndkbuild/libusb/msvc/errno.h File Reference . . . 833
34.270.1 Macro Definition Documentation . . . 833
EPERM . . . 833
ENOFILE . . . 833
ENOENT . . . 833
ESRCH . . . 833
EINTR . . . 833
EIO . . . 833
ENXIO . . . 833
E2BIG . . . 833
ENOEXEC . . . 833
EBADF . . . 833
ECHILD . . . 834
EAGAIN . . . 834
ENOMEM . . . 834
EACCES . . . 834
EFAULT . . . 834
EBUSY . . . 834
EEXIST . . . 834
EXDEV . . . 834
ENODEV . . . 834
ENOTDIR . . . 834
EISDIR . . . 834
EINVAL . . . 834
ENFILE . . . 834
EMFILE . . . 834
ENOTTY . . . 834
EFBIG . . . 834
ENOSPC . . . 834
ESPIPE . . . 834
EROFS . . . 834
EMLINK . . . 834
EPIPE . . . 834
EDOM . . . 834
ERANGE . . . 834
EDEADLOCK . . . 834
EDEADLK . . . 834
errno . . . 834
34.271 libusb/ndkbuild/libusb/msvc/inttypes.h File Reference . . . 834
34.271.1 Macro Definition Documentation . . . 837
PRId8 . . . 837

PRId16 . . . 837
PRId32 . . . 837
PRId64 . . . 837
PRIdLEAST8 . . . 837
PRIdLEAST16 . . . 837
PRIdLEAST32 . . . 837
PRIdLEAST64 . . . 837
PRIdFAST8 . . . 837
PRIdFAST16 . . . 837
PRIdFAST32 . . . 837
PRIdFAST64 . . . 837
PRIdMAX . . . 837
PRIi8 . . . 837
PRIi16 . . . 837
PRIi32 . . . 837
PRIi64 . . . 837
PRIiLEAST8 . . . 837
PRIiLEAST16 . . . 837
PRIiLEAST32 . . . 837
PRIiLEAST64 . . . 837
PRIiFAST8 . . . 837
PRIiFAST16 . . . 837
PRIiFAST32 . . . 837
PRIiFAST64 . . . 837
PRIiMAX . . . 837
PRIo8 . . . 837
PRIo16 . . . 837
PRIo32 . . . 837
PRIo64 . . . 837
PRIoLEAST8 . . . 837
PRIoLEAST16 . . . 837
PRIoLEAST32 . . . 837
PRIoLEAST64 . . . 837
PRIoFAST8 . . . 838
PRIoFAST16 . . . 838
PRIoFAST32 . . . 838
PRIoFAST64 . . . 838
PRIoMAX . . . 838
PRIu8 . . . 838
PRIu16 . . . 838
PRIu32 . . . 838
PRIu64 . . . 838
PRIuLEAST8 . . . 838
PRIuLEAST16 . . . 838
PRIuLEAST32 . . . 838
PRIuLEAST64 . . . 838
PRIuFAST8 . . . 838
PRIuFAST16 . . . 838
PRIuFAST32 . . . 838

PRIuFAST64 . . . 838
PRIuMAX . . . 838
PRIx8 . . . 838
PRIx16 . . . 838
PRIx32 . . . 838
PRIx64 . . . 838
PRIxLEAST8 . . . 838
PRIxLEAST16 . . . 838
PRIxLEAST32 . . . 838
PRIxLEAST64 . . . 838
PRIxFAST8 . . . 838
PRIxFAST16 . . . 838
PRIxFAST32 . . . 838
PRIxFAST64 . . . 838
PRIxMAX . . . 838
PRIX8 . . . 838
PRIX16 . . . 838
PRIX32 . . . 838
PRIX64 . . . 838
PRIXLEAST8 . . . 839
PRIXLEAST16 . . . 839
PRIXLEAST32 . . . 839
PRIXLEAST64 . . . 839
PRIXFAST8 . . . 839
PRIXFAST16 . . . 839
PRIXFAST32 . . . 839
PRIXFAST64 . . . 839
PRIXMAX . . . 839
SCNd16 . . . 839
SCNd32 . . . 839
SCNd64 . . . 839
SCNdLEAST16 . . . 839
SCNdLEAST32 . . . 839
SCNdLEAST64 . . . 839
SCNdFAST16 . . . 839
SCNdFAST32 . . . 839
SCNdFAST64 . . . 839
SCNdMAX . . . 839
SCNi16 . . . 839
SCNi32 . . . 839
SCNi64 . . . 839
SCNiLEAST16 . . . 839
SCNiLEAST32 . . . 839
SCNiLEAST64 . . . 839
SCNiFAST16 . . . 839
SCNiFAST32 . . . 839
SCNiFAST64 . . . 839
SCNiMAX . . . 839
SCNo16 . . . 839

SCNo32 . . . 839
SCNo64 . . . 839
SCNoLEAST16 . . . 839
SCNoLEAST32 . . . 839
SCNoLEAST64 . . . 839
SCNoFAST16 . . . 840
SCNoFAST32 . . . 840
SCNoFAST64 . . . 840
SCNoMAX . . . 840
SCNx16 . . . 840
SCNx32 . . . 840
SCNx64 . . . 840
SCNxLEAST16 . . . 840
SCNxLEAST32 . . . 840
SCNxLEAST64 . . . 840
SCNxFAST16 . . . 840
SCNxFAST32 . . . 840
SCNxFAST64 . . . 840
SCNxMAX . . . 840
SCNu16 . . . 840
SCNu32 . . . 840
SCNu64 . . . 840
SCNuLEAST16 . . . 840
SCNuLEAST32 . . . 840
SCNuLEAST64 . . . 840
SCNuFAST16 . . . 840
SCNuFAST32 . . . 840
SCNuFAST64 . . . 840
SCNuMAX . . . 840
PRIdPTR . . . 840
PRIiPTR . . . 840
PRIoPTR . . . 840
PRIuPTR . . . 840
PRIxPTR . . . 840
PRIXPTR . . . 840
SCNdPTR . . . 840
SCNiPTR . . . 840
SCNoPTR . . . 840
SCNxPTR . . . 840
SCNuPTR . . . 840
34.272 libusb/ndkbuild/libusb/msvc/missing.c File Reference . . . 841
34.272.1 Macro Definition Documentation . . . 841
ENVIRONMENT_REG_PATH . . . 841
34.272.2 Function Documentation . . . 841
getenv . . . 841
34.273 libusb/ndkbuild/libusb/msvc/missing.h File Reference . . . 841
34.273.1 Macro Definition Documentation . . . 841
SleepEx . . . 841
34.273.2 Function Documentation . . . 841

getenv . . . 841
34.274 lib/ndkbuild/libusb/msvc/stdint.h File Reference . . . 841
34.274.1 Macro Definition Documentation . . . 843
INT8_MIN . . . 843
INT16_MIN . . . 843
INT32_MIN . . . 843
INT64_MIN . . . 843
INT8_MAX . . . 843
INT16_MAX . . . 843
INT32_MAX . . . 843
INT64_MAX . . . 843
UINT8_MAX . . . 843
UINT16_MAX . . . 843
UINT32_MAX . . . 843
UINT64_MAX . . . 843
INT_LEAST8_MIN . . . 843
INT_LEAST16_MIN . . . 843
INT_LEAST32_MIN . . . 843
INT_LEAST64_MIN . . . 843
INT_LEAST8_MAX . . . 843
INT_LEAST16_MAX . . . 843
INT_LEAST32_MAX . . . 843
INT_LEAST64_MAX . . . 843
UINT_LEAST8_MAX . . . 843
UINT_LEAST16_MAX . . . 843
UINT_LEAST32_MAX . . . 843
UINT_LEAST64_MAX . . . 843
INT_FAST8_MIN . . . 843
INT_FAST16_MIN . . . 843
INT_FAST32_MIN . . . 843
INT_FAST64_MIN . . . 844
INT_FAST8_MAX . . . 844
INT_FAST16_MAX . . . 844
INT_FAST32_MAX . . . 844
INT_FAST64_MAX . . . 844
UINT_FAST8_MAX . . . 844
UINT_FAST16_MAX . . . 844
UINT_FAST32_MAX . . . 844
UINT_FAST64_MAX . . . 844
INTPTR_MIN . . . 844
INTPTR_MAX . . . 844
UINTPTR_MAX . . . 844
INTMAX_MIN . . . 844
INTMAX_MAX . . . 844
UINTMAX_MAX . . . 844
PTRDIFF_MIN . . . 844
PTRDIFF_MAX . . . 844
SIG_ATOMIC_MIN . . . 844
SIG_ATOMIC_MAX . . . 844

SIZE_MAX . . . 844
WCHAR_MIN . . . 844
WCHAR_MAX . . . 844
WINT_MIN . . . 844
WINT_MAX . . . 844
INT8_C . . . 844
INT16_C . . . 844
INT32_C . . . 844
INT64_C . . . 844
UINT8_C . . . 844
UINT16_C . . . 844
UINT32_C . . . 844
UINT64_C . . . 844
INTMAX_C . . . 844
UINTMAX_C . . . 844
34.274.2 Typedef Documentation . . . 844
intptr_t . . . 845
uintptr_t . . . 845
ptrdiff_t . . . 845
wchar_t . . . 845
wint_t . . . 845
wctype_t . . . 845
int8_t . . . 845
uint8_t . . . 845
int16_t . . . 845
uint16_t . . . 845
int32_t . . . 845
uint32_t . . . 845
int64_t . . . 845
uint64_t . . . 845
int_least8_t . . . 845
uint_least8_t . . . 845
int_least16_t . . . 845
uint_least16_t . . . 845
int_least32_t . . . 845
uint_least32_t . . . 845
int_least64_t . . . 845
uint_least64_t . . . 845
int_fast8_t . . . 845
uint_fast8_t . . . 845
int_fast16_t . . . 845
uint_fast16_t . . . 845
int_fast32_t . . . 845
uint_fast32_t . . . 845
int_fast64_t . . . 845
uint_fast64_t . . . 845
intmax_t . . . 845
uintmax_t . . . 845
34.275 /home/ndkbuild/libusb/tests/libusb_testlib.h File Reference . . . 845

34.275.1 Macro Definition Documentation . . . 846
bool . . . 846
true . . . 846
false . . . 846
LIBUSB_NULL_TEST . . . 846
34.275.2 Typedef Documentation . . . 846
libusb_testlib_test_function . . . 846
34.275.3 Enumeration Type Documentation . . . 846
libusb_testlib_result . . . 846
34.275.4 Function Documentation . . . 847
libusb_testlib_logf . . . 847
libusb_testlib_run_tests . . . 847
34.276 lib/ndkbuild/libusb/tests/stress.c File Reference . . . 847
34.276.1 Macro Definition Documentation . . . 847
LIST_COUNT . . . 847
34.276.2 Function Documentation . . . 847
main . . . 847
34.277 lib/ndkbuild/libusb/tests/testlib.c File Reference . . . 847
34.277.1 Macro Definition Documentation . . . 848
NULL_PATH . . . 848
INVALID_FD . . . 848
IGNORE_RETVAL . . . 848
34.277.2 Function Documentation . . . 848
libusb_testlib_logf . . . 848
libusb_testlib_run_tests . . . 848
34.278 lib/wrappers/scilab/foo.c File Reference . . . 848
34.278.1 Function Documentation . . . 848
foo_something . . . 848
34.279 lib/wrappers/scilab/foo.h File Reference . . . 848
34.279.1 Function Documentation . . . 848
foo_something . . . 848
34.280 samples/TestLib/aiocommon.c File Reference . . . 849
34.280.1 Macro Definition Documentation . . . 849
DUMP . . . 849
CNTS . . . 849
JCONF . . . 849
REPEAT . . . 849
34.280.2 Function Documentation . . . 849
get_channel_range . . . 849
process_aio_cmd_line . . . 849
print_aio_usage . . . 849
aio_list_devices . . . 849
aio_override_adcconfig_settings . . . 849
aio_supply_default_command_line_settings . . . 849
aio_override_aiobuf_settings . . . 849
34.280.3 Variable Documentation . . . 849
AIO_OPTIONS . . . 849
34.281 samples/TestLib/aiocommon.h File Reference . . . 850
34.281.1 Function Documentation . . . 850

get_channel_range . . . 850
process_aio_cmd_line . . . 850
print_aio_usage . . . 850
aio_list_devices . . . 850
aio_override_aiobuf_settings . . . 850
aio_override_adcconfig_settings . . . 850
aio_supply_default_command_line_settings . . . 850
34.281.2 Variable Documentation . . . 850
AIO_OPTIONS . . . 850
34.282 Samples/TestLib/TestCaseSetup.cpp File Reference . . . 850
34.282.1 Variable Documentation . . . 851
CURRENT_DEBUG_LEVEL . . . 851
34.283 Samples/TestLib/TestCaseSetup.h File Reference . . . 851
34.283.1 Macro Definition Documentation . . . 851
ERROR_LEVEL . . . 851
FATAL_LEVEL . . . 851
ALERT_LEVEL . . . 851
WARN_LEVEL . . . 851
INFO_LEVEL . . . 851
DEBUG_LEVEL . . . 851
TRACE_LEVEL . . . 851
LOG . . . 852
INFO . . . 852
TRACE . . . 852
DEBUG . . . 852
ERROR . . . 852
FATAL . . . 852
TERSE_LOGGING . . . 852
VERBOSE_LOGGING . . . 852
THROW_ERROR . . . 852
CHECK_RESULT . . . 852
34.283.2 Variable Documentation . . . 852
MAX_NAME_SIZE . . . 852
DEF_MAX_CHANNELS . . . 852
DEF_NUM_CHANNELS . . . 852
DEF_CAL_CHANNEL . . . 852
CURRENT_DEBUG_LEVEL . . . 852
34.284 Samples/USB-AI16-16/bulk_acquire_sample.c File Reference . . . 852
34.284.1 Function Documentation . . . 853
process_cmd_line . . . 853
main . . . 853
print_usage . . . 853
34.285 Samples/USB-AI16-16/burst_test.c File Reference . . . 853
34.285.1 Detailed Description . . . 853
34.285.2 Function Documentation . . . 854
find_ai_board . . . 854
34.286 Samples/USB-AI16-16/continuous_mode.c File Reference . . . 854
34.286.1 Detailed Description . . . 854
34.286.2 Function Documentation . . . 854

fnd . . . 854
34.287 Samples/USB-AI16-16/continuous_mode_callback.c File Reference . . . 854
34.287.1 Function Documentation . . . 855
get_channel_range . . . 855
process_cmd_line . . . 855
fnd . . . 855
capture_data . . . 855
main . . . 855
34.287.2 Variable Documentation . . . 855
fp . . . 855
34.288 Samples/USB-AI16-16/continuous_mode_from_json_config.c File Reference . . . 855
34.288.1 Detailed Description . . . 855
34.288.2 Function Documentation . . . 855
get_channel_range . . . 855
process_cmd_line . . . 855
fnd . . . 855
capture_data . . . 856
main . . . 856
34.288.3 Variable Documentation . . . 856
fp . . . 856
34.289 Samples/USB-AI16-16/daitest.cpp File Reference . . . 856
34.289.1 Function Documentation . . . 856
handle_signal . . . 856
main . . . 856
34.289.2 Variable Documentation . . . 856
exit_sample . . . 856
old_action . . . 856
34.290 Samples/USB-AI16-16/dio_sample.c File Reference . . . 856
34.290.1 Function Documentation . . . 856
find_ai_board . . . 856
main . . . 856
34.291 Samples/USB-AI16-16/diotest.c File Reference . . . 857
34.291.1 Function Documentation . . . 857
main . . . 857
34.292 Samples/USB-AI16-16/diotest2.cpp File Reference . . . 857
34.292.1 Function Documentation . . . 857
main . . . 857
34.293 Samples/USB-AI16-16/HOLD/dirktest.c File Reference . . . 857
34.293.1 Function Documentation . . . 857
get_channel_range . . . 857
process_cmd_line . . . 857
fnd . . . 857
capture_data . . . 858
main . . . 858
34.293.2 Variable Documentation . . . 858
fp . . . 858
34.294 Samples/USB-AI16-16/HOLD/julian_test.c File Reference . . . 858
34.294.1 Function Documentation . . . 858
find_ai_board . . . 858

main . . . 858
34.295 Samples/USB-AI16-16/HOLD/reverse_cal_table.cpp File Reference . . . 858
34.295.1 Function Documentation . . . 858
goDoIt . . . 858
main . . . 858
34.296 Samples/USB-AI16-16/reverse_cal_table.cpp File Reference . . . 858
34.296.1 Function Documentation . . . 859
goDoIt . . . 859
main . . . 859
34.297 Samples/USB-AI16-16/HOLD/sample_dio.c File Reference . . . 859
34.297.1 Function Documentation . . . 859
find_ai_board . . . 859
main . . . 859
34.298 Samples/USB-AI16-16/HOLD/slow_receiver_test.cpp File Reference . . . 859
34.298.1 Function Documentation . . . 859
process_cmd_line . . . 859
print_usage . . . 859
main . . . 859
34.299 Samples/USB-AI16-16/HOLD/test.c File Reference . . . 860
34.299.1 Function Documentation . . . 860
get_channel_range . . . 860
process_cmd_line . . . 860
fnd . . . 860
capture_data . . . 860
main . . . 860
34.299.2 Variable Documentation . . . 860
fp . . . 860
34.300 Samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp File Reference . . . 860
34.300.1 Detailed Description . . . 860
34.300.2 Function Documentation . . . 861
main . . . 861
34.301 Samples/USB-AI16-16/jni/read_channels_test.c File Reference . . . 861
34.301.1 Function Documentation . . . 861
find_ai_board . . . 861
main . . . 861
34.302 Samples/USB-AI16-16/read_channels_test.c File Reference . . . 862
34.302.1 Function Documentation . . . 862
find_ai_board . . . 862
main . . . 862
34.303 Samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp File Reference . . . 862
34.303.1 Function Documentation . . . 862
get_options . . . 862
main . . . 862
34.304 Samples/USB-AI16-16/sample.cpp File Reference . . . 863
34.304.1 Function Documentation . . . 863
main . . . 863
34.305 Samples/USB-AO16-16/sample.cpp File Reference . . . 863
34.305.1 Function Documentation . . . 863
main . . . 863

34.306 Samples/USB-DA12-8A/sample.cpp File Reference . . . 863
34.306.1 Function Documentation . . . 863
main . . . 863
34.307 Samples/USB-DIO-16/sample.cpp File Reference . . . 863
34.307.1 Function Documentation . . . 864
main . . . 864
34.308 Samples/USB-AI16-16/simp_test.cpp File Reference . . . 864
34.308.1 Function Documentation . . . 864
main . . . 864
34.309 Samples/USB-AI16-16/simple_continuous_with_json.c File Reference . . . 864
34.309.1 Detailed Description . . . 864
34.309.2 Function Documentation . . . 864
get_channel_range . . . 864
process_cmd_line . . . 865
fnd . . . 865
main . . . 865
34.309.3 Variable Documentation . . . 865
fp . . . 865
34.310 Samples/USB-AI16-16/start_stop_continuous.c File Reference . . . 865
34.310.1 Function Documentation . . . 865
get_channel_range . . . 865
process_cmd_line . . . 865
run_acquisition . . . 865
fnd . . . 865
main . . . 865
34.310.2 Variable Documentation . . . 865
fp . . . 865
34.311 Samples/USB-AI16-16/test_fastscan.cpp File Reference . . . 865
34.311.1 Function Documentation . . . 866
main . . . 866
34.311.2 Variable Documentation . . . 866
CURRENT_DEBUG_LEVEL . . . 866
34.312 Samples/USB-ARB1/stream_test.c File Reference . . . 866
34.312.1 Function Documentation . . . 866
find_ai_board . . . 866
main . . . 866
34.313 Samples/USB-DA12-8A/SampleClass.cpp File Reference . . . 866
34.313.1 Function Documentation . . . 866
main . . . 866
34.314 Samples/USB-DIO-16/receiver.cpp File Reference . . . 866
34.314.1 Function Documentation . . . 867
main . . . 867
34.315 Samples/USB-DIO-16/standalone_receiver.c File Reference . . . 867
34.315.1 Function Documentation . . . 867
main . . . 867
34.316 Samples/USB-DIO-32/daisample.c File Reference . . . 867
34.316.1 Function Documentation . . . 867
main . . . 867
34.317 Samples/USB-DIO-32/read_and_write_sample.c File Reference . . . 867

34.317.1 Function Documentation . . . 867
main . . . 867
34.318 Samples/USB-DIO-96/read_and_write_sample.c File Reference . . . 868
34.318.1 Macro Definition Documentation . . . 869
BITS_PER_BYTE . . . 869
DEVICES_REQUIRED . . . 869
MAX_DIO_BYTES . . . 869
MASK_BYTES . . . 869
MAX_NAME_SIZE . . . 869
34.318.2 Enumeration Type Documentation . . . 869
EXIT_CODE . . . 869
34.318.3 Function Documentation . . . 869
find_dio_96 . . . 869
main . . . 869
34.319 Samples/USB-DIO-32/sample3.c File Reference . . . 869
34.319.1 Function Documentation . . . 870
main . . . 870
34.320 Samples/USB-DIO-48/read_and_write.c File Reference . . . 870
34.320.1 Detailed Description . . . 870
34.320.2 Function Documentation . . . 871
find_dio . . . 871
CHECK_RESULT . . . 871
main . . . 871
34.321 Samples/USB-DIO-96/dio96_read_write.c File Reference . . . 871
34.321.1 Macro Definition Documentation . . . 871
BITS_PER_BYTE . . . 871
DEVICES_REQUIRED . . . 871
MAX_DIO_BYTES . . . 871
MASK_BYTES . . . 871
MAX_NAME_SIZE . . . 871
PORT_C . . . 871
PORT_B . . . 871
PORT_A . . . 871
MAKE_MASK . . . 871
34.321.2 Function Documentation . . . 872
find_dio_96 . . . 872
show_byte . . . 872
main . . . 872
34.322 Samples/USB-DIO-96/mytest.c File Reference . . . 872
34.322.1 Macro Definition Documentation . . . 873
BITS_PER_BYTE . . . 873
DEVICES_REQUIRED . . . 873
MAX_DIO_BYTES . . . 873
MASK_BYTES . . . 873
MAX_NAME_SIZE . . . 873
PORT_C . . . 873
PORT_B . . . 873
PORT_A . . . 873
MAKE_MASK . . . 873

34.322.2 Function Documentation . . . 873
find_dio_96 . . . 873
show_byte . . . 874
main . . . 874
34.323 Samples/USB-DIO-96/tmp.c File Reference . . . 874
34.323.1 Macro Definition Documentation . . . 874
BITS_PER_BYTE . . . 874
DEVICES_REQUIRED . . . 874
MAX_DIO_BYTES . . . 874
MASK_BYTES . . . 874
MAX_NAME_SIZE . . . 874
PORT_C . . . 874
PORT_B . . . 875
PORT_A . . . 875
MAKE_MASK . . . 875
34.323.2 Function Documentation . . . 875
find_dio_96 . . . 875
show_byte . . . 875
main . . . 875
34.324 Samples/USB-DIO-96/write_sample.c File Reference . . . 875
34.324.1 Macro Definition Documentation . . . 876
BITS_PER_BYTE . . . 876
DEVICES_REQUIRED . . . 876
MAX_DIO_BYTES . . . 876
MASK_BYTES . . . 876
MAX_NAME_SIZE . . . 876
34.324.2 Enumeration Type Documentation . . . 876
EXIT_CODE . . . 876
34.324.3 Function Documentation . . . 877
find_dio_96 . . . 877
fnd . . . 877
main . . . 877
34.325 Samples/USB-IDIO-16_8/idio_sample.c File Reference . . . 877
34.325.1 Detailed Description . . . 877
34.325.2 Macro Definition Documentation . . . 878
RATE_LIMIT . . . 878
34.325.3 Function Documentation . . . 878
main . . . 878
34.326 Samples/USB-IDIO-16_8/idio_sample2.c File Reference . . . 878
34.326.1 Detailed Description . . . 878
34.326.2 Macro Definition Documentation . . . 878
RATE_LIMIT . . . 878
34.326.3 Function Documentation . . . 879
main . . . 879
34.327 Samples/USB-IDIO-16_8/perftest.c File Reference . . . 879
34.327.1 Function Documentation . . . 879
main . . . 879
34.328 Samples/USB-IIRO-16_8/iiro_sample.c File Reference . . . 879
34.328.1 Detailed Description . . . 880

34.328.2 Macro Definition Documentation . . . 880
RATE_LIMIT . . . 880
34.328.3 Function Documentation . . . 880
find_idio . . . 880
main . . . 880
34.329 media/jdamon/Development/Documents/Projects/aiousb_patrick_mcbride_issue/AIOUSB/README.md File Reference . . . 880
34.330 lib/ndkbuild/libusb/README.md File Reference . . . 880
34.331 lib/wrappers/README.md File Reference . . . 880
34.332 samples/USB-AI16-16/android/read_channels_test/README.md File Reference . . . 880
34.333 samples/USB-AI16-16/android/README.md File Reference . . . 880
34.334 samples/USB-AI16-16/java/extcal/README.md File Reference . . . 880
34.335 samples/USB-AI16-16/java/read_channels_test/native-utils/README.md File Reference . . . 880
34.336 samples/USB-AI16-16/java/read_channels_test/README.md File Reference . . . 880
34.337 samples/USB-AI16-16/java/README.md File Reference . . . 880

Index 881

Chapter 1

Main Page

ACCES I/O Products, Inc.

1.1 AIOUSB driver library

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

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

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

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

1.1.1 Prerequisites

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

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

Ubuntu / Debian

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

Fedora / Red Hat

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

Open SUSE

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

Mac OS X

Homebrew

```
brew install libusb  cmake
```

Darwin Ports

```
sudo port install libusb cmake
```

Rasberry Pi

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

Beagle Board

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

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

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

1.2.1 GNU make build

You will need to do the following

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

1.2.2 Build with CMake

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

1.2.3 Installation

Linux Installation

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

Mac Installation (work in progress!!)

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

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

Windows Installation

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

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

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

2. Download the `WinUSB 8.10 drivers` and extract them to a saved directory.

3. In Device Manager right click on the original USB Data acquisition device that should now have no driver associated with it and it should be listed as a generic USB device. Right click on the device and click "Update Driver" and instruct Windows to look for the drivers in the "Saved_Directory" from step 2.

4. After it has installed, under Device Manager the device should now be listed as a Data Acquisition product. In addition, Check the name the device ended up with; it should have a (WinUSB) tag, like "ACCES USB-IIRO-16 (WinUSB)". If it has a (CyUSB) tag instead, something went wrong, please tell us.

5. Make sure that you already have a full Cygwin installation along with the following packages:

   - python
   - python-development
   - cmake
   - gnumake
   - libusb-1.0
   - gcc
   - g++
   - swig (if you want to build wrapper language support)

6. Follow the instructions listed at either `Cmake build` or `GNU make`

Extra Language Support

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

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

1.2.4 How to build Wrapper languages

CMake

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

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

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

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

or if you have installed cmake-gui, then

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

Regular Make system for building wrapper language support

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

Perl

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

Java

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

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

Python

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

Ruby

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

Octave

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

R

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

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

Sincerely,

The ACCES I/O Development team.

Chapter 2

Compiling and Installation

2.1 Compiling from source on Linux/MacOS X / UNIX

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

- Build with CMake
- GNU make build

2.2 Compiling from source on Windows

TBD

Todo Complete the Windows port of the AIOUSB libraries

2.3 Installing the Compiled libraries

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

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

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

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

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

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

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

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

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

Chapter 3

AIOUSB C libary

3.1 Overview

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

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

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

3.2 Other stuff

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

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

export AIOUSBLIBDIR="path to the AIOUSB libraries"

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

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

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

3.3 Compiling sample

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

Chapter 4

AIOUSB C++ Class library

4.1 Introduction

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

4.2 Packaging

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

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

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

4.3 Deprecated

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

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

language other than C or C++ we provide a wrappers directory that includes support for Java, Perl, Python, Ruby, R , Php, Octave and Matlab.

We apologize for any inconvenience this might have caused,

Sincerely,

The ACCES I/O Products Software team.

Chapter 5

C/C++ Samples

5.1 Overview

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

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

USB-AI16-16 samples

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

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

USB-DIO-32 samples

USB-DIO-16 samples

USB-AO16-16 samples

USB-DA12-8A samples

5.3 USB-AI16-16

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

extcal.c External calibration sample

continuous_mode.c Continuous Mode Samples

continuous_mode_from_json_config.c Continuous Mode sample usng a JSON config file

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

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

5.3.1 extcal.c

Extcal

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

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

Todo Setup BUILDING Tag

5.3.2 continuous_mode.c

Continuous Mode

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

The key steps for running a continuous mode acquisition are:

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

Todo Reference building tag

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

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

```
ADConfigBlock configBlock;

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

ADC_QueryCal( AIOContinuousBuf_GetDeviceIndex(buf) );

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

5.3.3 continuous_mode_from_json_config.c

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

5.3.4 burst_test.c

Overview

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

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

Parts of the sample

Command line parsing

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

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

Todo Document the Command line Parsing helper library

Create a new AIOContinuousBuf

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

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

Set the device index for the AIOContinuousBuf

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

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

Initialize the AIOContinuousBuf

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

```
AIOContinuousBufInitConfiguration( buf );
```

Set the Clock rate / Data Acquisition speed

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

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

Start the continuous mode callback

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

```
AIOContinuousBufInitiateCallbackAcquisition(buf);
```

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

Acquire data until completed

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

```
while ( AIOContinuousBufPending(buf) ) {

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

Do something with the data

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

5.3.5 bulk_aquire_test.c

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

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

Overview

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

idio_sample.c Basic sample for USB-IDIO products

idio_sample2.c Sample program demonstrating stuff

Build

5.4.1 idio_sample.c

Sample2

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

Todo Complete this example

5.4.2 idio_sample2.c

Sample2

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

Todo Complete this example

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

Overview

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

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

Build

5.5.1 iiro_sample.c

iiro_sample

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

Todo Complete this example

5.6 USB-DIO-32

USB-DIO-32 Sample Program Release Notes

5.6.1 Overview

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

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

5.6.2 C/C++ Language Sample

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

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

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

Building

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

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

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

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

Executing

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

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

```
USB-DIO-32 sample program version 1.17, 26 November 2009
  AIOUSB library version 1.84, 22 December 2009
  This program demonstrates communicating with 2 USB-DIO-32 devices on
  the same USB bus. For simplicity, it uses the first 2 such devices
  found on the bus.
ACCES devices found:
  Device at index 0:
    Product ID: 0x8040
    Product name: USB-AI16-16A
    Number of digital I/O bytes: 2
    Number of counters: 1
  Device at index 1:
    Product ID: 0x8001
    Product name: USB-DIO-32
    Number of digital I/O bytes: 4
    Number of counters: 3
  Device at index 2:
    Product ID: 0x8001
```

```
    Product name: USB-DIO-32
    Number of digital I/O bytes: 4
    Number of counters: 3
Serial number of device at index 1: 40e391cdff3dd1bb
Serial number of device at index 2: 40e391cdf95aa30c
Device at index 1 successfully configured
Device at index 2 successfully configured
Read the following values from device at index 1: 0x11 0x22 0x33 0x44 (correct)
Read the following values from device at index 2: 0x66 0x65 0x64 0x63 (correct)
Writing patterns to devices: 0 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0
 0xb0 0xc0 0xd0 0xe0 0xf0
All patterns written were read back correctly
```

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

5.6.3 Java Sample

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

Building

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

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

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

Executing

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

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

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

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

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

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

```
USB-DIO-32 sample program version: 1.3, 25 December 2009
  AIOUSB Java library version: 1.6, 17 December 2009
  AIOUSB library version: 1.84, 22 December 2009
  JRE version: 1.6.0_17
  OS version: Linux amd64 2.6.31.5-0.1-custom
  This program demonstrates controlling a USB-DIO-32 device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus.
ACCES devices found:
  Device at index 0
    Product ID: 0x8040
    Product name: USB-AI16-16A
    Serial number: 0x40e38f15d5c94894
    Number of A/D channels: 16
    Number of MUXed A/D channels: 16
    Number of digital I/O ports: 2
    Number of digital I/O channels: 16
    Number of tristate groups: 0
    Number of tristate channels: 0
    Number of counter blocks: 1
    Number of counters: 3
  Device at index 1
    Product ID: 0x8001
    Product name: USB-DIO-32
    Serial number: 0x40e39acf9e8dd7cc
    Number of digital I/O ports: 4
    Number of digital I/O channels: 32
    Number of tristate groups: 0
    Number of tristate channels: 0
    Number of counter blocks: 3
    Number of counters: 9
EEPROM contents:
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

```
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
Configuring digital I/O ... successful
Turning all outputs on: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
      31
Turning all outputs off: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
       31
Turning all outputs on ... successful
Turning all outputs off ... successful
```

5.7 USB-DIO-16

USB-DIO-16A Sample Program Release Notes

5.7.1 Overview

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

Basic AIOUSB Sample - sample.cpp, receiver.cpp

AIOUSB Java Sample - Sample.java

5.7.2 Basic AIOUSB Sample

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

- Initializing and shutting down the API – AIOUSB_Init(), AIOUSB_Exit()
- Identifying devices on the USB bus – QueryDeviceInfo()
- Obtaining the serial number of a device on the bus – GetDeviceSerialNumber()
- Configuring the board – DIO_ConfigureEx()
- Writing to, and reading from a digital I/O stream – DIO_StreamSetClocks(), DIO_StreamOpen(), DIO_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 -laiousb -lusb-1.0 -o sample g++ receiver.cpp -laiousb -lusb-1.0 -o receiver

5.7.4 Executing

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

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

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

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

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

5.7.5 AIOUSB Java Sample

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

Building

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

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

Executing

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

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

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

or

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

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

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

5.8 USB-AO16-16

USB-AO16-16A Sample Program Release Notes

5.8.1 Overview

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

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

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

Building

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

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

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

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

Executing

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

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

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

5.9 USB-DA12-8A

USB-DA12-8A Sample Program Release Notes

5.9.1 Overview

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

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

5.9.2 AIOUSB C Sample

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

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

Building

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

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

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

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

Executing

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

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

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

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

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

5.9.3 AIOUSB C++ Sample

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

Building

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

Executing

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

./SampleClass

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

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

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

5.9.4 AIOUSB Java Sample

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

Building

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

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

Executing

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

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

or

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

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

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

Chapter 6

Wrappers

6.1 Overview

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

Building Wrappers

Todo Complete Wrapper Doxygen page

6.2 Building Wrappers

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

1. Setup build variables

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

1. Make sure that you have build libaiousb∗ . To do this you should have run "make" in the directories $AIOUSB_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 libaiousb∗ . 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 modified similarly.

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

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

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

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

7.5 Copy Udev Rules File To System Directory

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

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

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

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

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

7.6 Troubleshooting The Udev Rules File

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

7.7 Manually Uploading Firmware to USB Devices

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

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

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

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

7.8 Minimum Required Files

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

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

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

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

Chapter 8

LIBUSB Overview

8.1 Overview

Somet stuff

8.2 Libusb Other Stuff

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

8.3 Compiling sample

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

or

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

Chapter 9

Caveats

9.1 Device resets

The libusb_reset_device() function allows you to reset a device. If your program has to call such a function, it should obviously be aware that the reset will cause device state to change (e.g. register values may be reset).

The problem is that any other program could reset the device your program is working with, at any time. libusb does not offer a mechanism to inform you when this has happened, so if someone else resets your device it will not be clear to your own program why the device state has changed.

Ultimately, this is a limitation of writing drivers in userspace. Separation from the USB stack in the underlying kernel makes it difficult for the operating system to deliver such notifications to your program. The Linux kernel USB stack allows such reset notifications to be delivered to in-kernel USB drivers, but it is not clear how such notifications could be delivered to second-class drivers that live in userspace.

9.2 Blocking-only functionality

The functionality listed below is only available through synchronous, blocking functions. There are no asynchronous/non-blocking alternatives, and no clear ways of implementing these.

- Configuration activation (libusb_set_configuration())
- Interface/alternate setting activation (libusb_set_interface_alt_setting())
- Releasing of interfaces (libusb_release_interface())
- Clearing of halt/stall condition (libusb_clear_halt())
- Device resets (libusb_reset_device())

9.3 Configuration selection and handling

When libusb presents a device handle to an application, there is a chance that the corresponding device may be in unconfigured state. For devices with multiple configurations, there is also a chance that the configuration currently selected is not the one that the application wants to use.

The obvious solution is to add a call to libusb_set_configuration() early on during your device initialization routines, but there are caveats to be aware of:

1. If the device is already in the desired configuration, calling libusb_set_configuration() using the same configuration value will cause a lightweight device reset. This may not be desirable behaviour.
2. In the case where the desired configuration is already active, libusb may not even be able to perform a lightweight device reset. For example, take my USB keyboard with fingerprint reader: I'm interested in driving the fingerprint reader interface through libusb, but the kernel's USB-HID driver will almost always have claimed the keyboard interface. Because the kernel has claimed an interface, it is not even possible to perform the lightweight device reset, so libusb_set_configuration() will fail. (Luckily the device in question only has a single configuration.)
3. libusb will be unable to set a configuration if other programs or drivers have claimed interfaces. In particular, this means that kernel drivers must be detached from all the interfaces before libusb_set_configuration() may succeed.

One solution to some of the above problems is to consider the currently active configuration. If the configuration we want is already active, then we don't have to select any configuration:

```
cfg = -1;
libusb_get_configuration(dev, &cfg);
if (cfg != desired)
        libusb_set_configuration(dev, desired);
```

This is probably suitable for most scenarios, but is inherently racy: another application or driver may change the selected configuration *after* the libusb_get_configuration() call.

Even in cases where libusb_set_configuration() succeeds, consider that other applications or drivers may change configuration after your application calls libusb_set_configuration().

One possible way to lock your device into a specific configuration is as follows:

1. Set the desired configuration (or use the logic above to realise that it is already in the desired configuration)
2. Claim the interface that you wish to use
3. Check that the currently active configuration is the one that you want to use.

The above method works because once an interface is claimed, no application or driver is able to select another configuration.

9.4 Early transfer completion

NOTE: This section is currently Linux-centric. I am not sure if any of these considerations apply to Darwin or other platforms.

When a transfer completes early (i.e. when less data is received/sent in any one packet than the transfer buffer allows for) then libusb is designed to terminate the transfer immediately, not transferring or receiving any more data unless other transfers have been queued by the user.

On legacy platforms, libusb is unable to do this in all situations. After the incomplete packet occurs, "surplus" data may be transferred. For recent versions of libusb, this information is kept (the data length of the transfer is updated) and, for device-to-host transfers, any surplus data was added to the buffer. Still, this is not a nice solution because it loses the information about the end of the short packet, and the user probably wanted that surplus data to arrive in the next logical transfer.

9.5 Zero length packets

- libusb is able to send a packet of zero length to an endpoint simply by submitting a transfer of zero length.
- The LIBUSB_TRANSFER_ADD_ZERO_PACKET flag is currently only supported on Linux.

Chapter 10

Contexts

It is possible that libusb may be used simultaneously from two independent libraries linked into the same executable.

For example, if your application has a plugin-like system which allows the user to dynamically load a range of modules into your program, it is feasible that two independently developed modules may both use libusb.

libusb is written to allow for these multiple user scenarios. The two "instances" of libusb will not interfere: libusb_set_debug() calls from one user will not affect the same settings for other users, other users can continue using libusb after one of them calls libusb_exit(), etc.

This is made possible through libusb's *context* concept. When you call libusb_init(), you are (optionally) given a context. You can then pass this context pointer back into future libusb functions.

In order to keep things simple for more simplistic applications, it is legal to pass NULL to all functions requiring a context pointer (as long as you're sure no other code will attempt to use libusb from the same process). When you pass NULL, the default context will be used. The default context is created the first time a process calls libusb_init() when no other context is alive. Contexts are destroyed during libusb_exit().

The default context is reference-counted and can be shared. That means that if libusb_init(NULL) is called twice within the same process, the two users end up sharing the same context. The deinitialization and freeing of the default context will only happen when the last user calls libusb_exit(). In other words, the default context is created and initialized when its reference count goes from 0 to 1, and is deinitialized and destroyed when its reference count goes from 1 to 0.

You may be wondering why only a subset of libusb functions require a context pointer in their function definition. Internally, libusb stores context pointers in other objects (e.g. libusb_device instances) and hence can infer the context from those objects.

Chapter 11

Application Programming Interface

This is the complete list of libusb functions, structures and enumerations in alphabetical order.

11.1 Functions

- libusb_alloc_streams()
- libusb_alloc_transfer()
- libusb_attach_kernel_driver()
- libusb_bulk_transfer()
- libusb_cancel_transfer()
- libusb_claim_interface()
- libusb_clear_halt()
- libusb_close()
- libusb_control_transfer()
- libusb_control_transfer_get_data()
- libusb_control_transfer_get_setup()
- libusb_cpu_to_le16()
- libusb_detach_kernel_driver()
- libusb_dev_mem_alloc()
- libusb_dev_mem_free()
- libusb_error_name()
- libusb_event_handler_active()
- libusb_event_handling_ok()
- libusb_exit()
- libusb_fill_bulk_stream_transfer()
- libusb_fill_bulk_transfer()
- libusb_fill_control_setup()
- libusb_fill_control_transfer()
- libusb_fill_interrupt_transfer()
- libusb_fill_iso_transfer()
- libusb_free_bos_descriptor()
- libusb_free_config_descriptor()
- libusb_free_container_id_descriptor()
- libusb_free_device_list()
- libusb_free_pollfds()
- libusb_free_ss_endpoint_companion_descriptor()

- libusb_free_ss_usb_device_capability_descriptor()
- libusb_free_streams()
- libusb_free_transfer()
- libusb_free_usb_2_0_extension_descriptor()
- libusb_get_active_config_descriptor()
- libusb_get_bos_descriptor()
- libusb_get_bus_number()
- libusb_get_config_descriptor()
- libusb_get_config_descriptor_by_value()
- libusb_get_configuration()
- libusb_get_container_id_descriptor()
- libusb_get_descriptor()
- libusb_get_device()
- libusb_get_device_address()
- libusb_get_device_descriptor()
- libusb_get_device_list()
- libusb_get_device_speed()
- libusb_get_iso_packet_buffer()
- libusb_get_iso_packet_buffer_simple()
- libusb_get_max_iso_packet_size()
- libusb_get_max_packet_size()
- libusb_get_next_timeout()
- libusb_get_parent()
- libusb_get_pollfds()
- libusb_get_port_number()
- libusb_get_port_numbers()
- libusb_get_port_path()
- libusb_get_ss_endpoint_companion_descriptor()
- libusb_get_ss_usb_device_capability_descriptor()
- libusb_get_string_descriptor()
- libusb_get_string_descriptor_ascii()
- libusb_get_usb_2_0_extension_descriptor()
- libusb_get_version()
- libusb_handle_events()
- libusb_handle_events_completed()
- libusb_handle_events_locked()
- libusb_handle_events_timeout()
- libusb_handle_events_timeout_completed()
- libusb_has_capability()
- libusb_hotplug_deregister_callback()
- libusb_hotplug_register_callback()
- libusb_init()
- libusb_interrupt_event_handler()
- libusb_interrupt_transfer()

- libusb_kernel_driver_active()
- libusb_lock_events()
- libusb_lock_event_waiters()
- libusb_open()
- libusb_open_device_with_vid_pid()
- libusb_pollfds_handle_timeouts()
- libusb_ref_device()
- libusb_release_interface()
- libusb_reset_device()
- libusb_set_auto_detach_kernel_driver()
- libusb_set_configuration()
- libusb_set_debug()
- libusb_set_interface_alt_setting()
- libusb_set_iso_packet_lengths()
- libusb_setlocale()
- libusb_set_pollfd_notifiers()
- libusb_strerror()
- libusb_submit_transfer()
- libusb_transfer_get_stream_id()
- libusb_transfer_set_stream_id()
- libusb_try_lock_events()
- libusb_unlock_events()
- libusb_unlock_event_waiters()
- libusb_unref_device()
- libusb_wait_for_event()

11.2 Structures

- libusb_bos_descriptor
- libusb_bos_dev_capability_descriptor
- libusb_config_descriptor
- libusb_container_id_descriptor
- libusb_context
- libusb_control_setup
- libusb_device
- libusb_device_descriptor
- libusb_device_handle
- libusb_endpoint_descriptor
- libusb_interface
- libusb_interface_descriptor
- libusb_iso_packet_descriptor
- libusb_pollfd
- libusb_ss_endpoint_companion_descriptor
- libusb_ss_usb_device_capability_descriptor
- libusb_transfer
- libusb_usb_2_0_extension_descriptor
- libusb_version

11.3 Enums

- libusb_bos_type
- libusb_capability
- libusb_class_code
- libusb_descriptor_type
- libusb_endpoint_direction
- libusb_error
- libusb_iso_sync_type
- libusb_iso_usage_type
- libusb_log_level
- libusb_request_recipient
- libusb_request_type
- libusb_speed
- libusb_ss_usb_device_capability_attributes
- libusb_standard_request
- libusb_supported_speed
- libusb_transfer_flags
- libusb_transfer_status
- libusb_transfer_type
- libusb_usb_2_0_extension_attributes

Chapter 12

Device hotplug event notification

12.1 Introduction

Version 1.0.16, LIBUSB_API_VERSION >= 0x01000102, has added support for hotplug events on **some** platforms (you should test if your platform supports hotplug notification by calling libusb_has_capability() with parameter LIBUSB_CAP_HAS_HOTPLUG).

This interface allows you to request notification for the arrival and departure of matching USB devices.

To receive hotplug notification you register a callback by calling libusb_hotplug_register_callback(). This function will optionally return a callback handle that can be passed to libusb_hotplug_deregister_callback().

A callback function must return an int (0 or 1) indicating whether the callback is expecting additional events. Returning 0 will rearm the callback and 1 will cause the callback to be deregistered. Note that when callbacks are called from libusb_hotplug_register_callback() because of the LIBUSB_HOTPLUG_ENUMERATE flag, the callback return value is ignored, iow you cannot cause a callback to be deregistered by returning 1 when it is called from libusb_hotplug_register_callback().

Callbacks for a particular context are automatically deregistered by libusb_exit().

As of 1.0.16 there are two supported hotplug events:

- LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED: A device has arrived and is ready to use
- LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT: A device has left and is no longer available

A hotplug event can listen for either or both of these events.

Note: If you receive notification that a device has left and you have any a libusb_device_handles for the device it is up to you to call libusb_close() on each device handle to free up any remaining resources associated with the device. Once a device has left any libusb_device_handle associated with the device are invalid and will remain so even if the device comes back.

When handling a LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED event it is considered safe to call any libusb function that takes a libusb_device. It also safe to open a device and submit asynchronous transfers. However, most other functions that take a libusb_device_handle are **not** safe to call. Examples of such functions are any of the synchronous API functions or the blocking functions that retrieve various USB descriptors. These functions must be used outside of the context of the hotplug callback.

When handling a LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT event the only safe function is libusb_get_device_descriptor().

The following code provides an example of the usage of the hotplug interface:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <libusb.h>

static int count = 0;

int hotplug_callback(struct libusb_context *ctx, struct
      libusb_device *dev,
                     libusb_hotplug_event event, void *user_data) {
  static libusb_device_handle *dev_handle = NULL;
  struct libusb_device_descriptor desc;
  int rc;

  (void)libusb_get_device_descriptor(dev, &desc);

  if (LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED == event) {
    rc = libusb_open(dev, &dev_handle);
    if (LIBUSB_SUCCESS != rc) {
      printf("Could not open USB device\n");
    }
  } else if (LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT == event) {
    if (dev_handle) {
      libusb_close(dev_handle);
      dev_handle = NULL;
    }
```

```
  } else {
    printf("Unhandled event %d\n", event);
  }
  count++;

  return 0;
}

int main (void) {
  libusb_hotplug_callback_handle callback_handle;
  int rc;

  libusb_init(NULL);

  rc = libusb_hotplug_register_callback(NULL,
      LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED |
                                          LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT, 0
      , 0x045a, 0x5005,
                                          LIBUSB_HOTPLUG_MATCH_ANY, hotplug_callback,
       NULL,
                                          &callback_handle);
  if (LIBUSB_SUCCESS != rc) {
    printf("Error creating a hotplug callback\n");
    libusb_exit(NULL);
    return EXIT_FAILURE;
  }

  while (count < 2) {
    libusb_handle_events_completed(NULL, NULL);
    nanosleep(&(struct timespec){0, 10000000UL}, NULL);
  }

  libusb_hotplug_deregister_callback(NULL, callback_handle);
  libusb_exit(NULL);

  return 0;
}
```

Chapter 13

Synchronous and asynchronous device I/O

13.1 Introduction

If you're using libusb in your application, you're probably wanting to perform I/O with devices - you want to perform USB data transfers.

libusb offers two separate interfaces for device I/O. This page aims to introduce the two in order to help you decide which one is more suitable for your application. You can also choose to use both interfaces in your application by considering each transfer on a case-by-case basis.

Once you have read through the following discussion, you should consult the detailed API documentation pages for the details:

- Synchronous device I/O
- Asynchronous device I/O

13.2 Transfers at a logical level

At a logical level, USB transfers typically happen in two parts. For example, when reading data from a endpoint:

1. A request for data is sent to the device
2. Some time later, the incoming data is received by the host

or when writing data to an endpoint:

1. The data is sent to the device
2. Some time later, the host receives acknowledgement from the device that the data has been transferred.

There may be an indefinite delay between the two steps. Consider a fictional USB input device with a button that the user can press. In order to determine when the button is pressed, you would likely submit a request to read data on a bulk or interrupt endpoint and wait for data to arrive. Data will arrive when the button is pressed by the user, which is potentially hours later.

libusb offers both a synchronous and an asynchronous interface to performing USB transfers. The main difference is that the synchronous interface combines both steps indicated above into a single function call, whereas the asynchronous interface separates them.

13.3 The synchronous interface

The synchronous I/O interface allows you to perform a USB transfer with a single function call. When the function call returns, the transfer has completed and you can parse the results.

If you have used the libusb-0.1 before, this I/O style will seem familar to you. libusb-0.1 only offered a synchronous interface.

In our input device example, to read button presses you might write code in the following style:

```
unsigned char data[4];
int actual_length;
int r = libusb_bulk_transfer(dev_handle, LIBUSB_ENDPOINT_IN, data,
      sizeof(data), &actual_length, 0);
if (r == 0 && actual_length == sizeof(data)) {
        // results of the transaction can now be found in the data buffer
        // parse them here and report button press
} else {
        error();
}
```

The main advantage of this model is simplicity: you did everything with a single simple function call.

However, this interface has its limitations. Your application will sleep inside libusb_bulk_transfer() until the transaction has completed. If it takes the user 3 hours to press the button, your application will be sleeping for that long. Execution will be tied up inside the library - the entire thread will be useless for that duration.

Another issue is that by tieing up the thread with that single transaction there is no possibility of performing I/O with multiple endpoints and/or multiple devices simultaneously, unless you resort to creating one thread per transaction.

Additionally, there is no opportunity to cancel the transfer after the request has been submitted.

For details on how to use the synchronous API, see the synchronous I/O API documentation pages.

13.4 The asynchronous interface

Asynchronous I/O is the most significant new feature in libusb-1.0. Although it is a more complex interface, it solves all the issues detailed above.

Instead of providing which functions that block until the I/O has complete, libusb's asynchronous interface presents non-blocking functions which begin a transfer and then return immediately. Your application passes a callback function pointer to this non-blocking function, which libusb will call with the results of the transaction when it has completed.

Transfers which have been submitted through the non-blocking functions can be cancelled with a separate function call.

The non-blocking nature of this interface allows you to be simultaneously performing I/O to multiple endpoints on multiple devices, without having to use threads.

This added flexibility does come with some complications though:

- In the interest of being a lightweight library, libusb does not create threads and can only operate when your application is calling into it. Your application must call into libusb from it's main loop when events are ready to be handled, or you must use some other scheme to allow libusb to undertake whatever work needs to be done.
- libusb also needs to be called into at certain fixed points in time in order to accurately handle transfer timeouts.
- Memory handling becomes more complex. You cannot use stack memory unless the function with that stack is guaranteed not to return until the transfer callback has finished executing.
- You generally lose some linearity from your code flow because submitting the transfer request is done in a separate function from where the transfer results are handled. This becomes particularly obvious when you want to submit a second transfer based on the results of an earlier transfer.

Internally, libusb's synchronous interface is expressed in terms of function calls to the asynchronous interface.

For details on how to use the asynchronous API, see the asynchronous I/O API documentation pages.

Chapter 14

Packets and overflows

14.1 Packet abstraction

The USB specifications describe how data is transmitted in packets, with constraints on packet size defined by endpoint descriptors. The host must not send data payloads larger than the endpoint's maximum packet size.

libusb and the underlying OS abstract out the packet concept, allowing you to request transfers of any size. Internally, the request will be divided up into correctly-sized packets. You do not have to be concerned with packet sizes, but there is one exception when considering overflows.

14.2 Bulk/interrupt transfer overflows

When requesting data on a bulk endpoint, libusb requires you to supply a buffer and the maximum number of bytes of data that libusb can put in that buffer. However, the size of the buffer is not communicated to the device - the device is just asked to send any amount of data.

There is no problem if the device sends an amount of data that is less than or equal to the buffer size. libusb reports this condition to you through the libusb_transfer.actual_length field.

Problems may occur if the device attempts to send more data than can fit in the buffer. libusb reports LIBUSB_TRANSFER_OVERFLOW for this condition but other behaviour is largely undefined: actual_length may or may not be accurate, the chunk of data that can fit in the buffer (before overflow) may or may not have been transferred.

Overflows are nasty, but can be avoided. Even though you were told to ignore packets above, think about the lower level details: each transfer is split into packets (typically small, with a maximum size of 512 bytes). Overflows can only happen if the final packet in an incoming data transfer is smaller than the actual packet that the device wants to transfer. Therefore, you will never see an overflow if your transfer buffer size is a multiple of the endpoint's packet size: the final packet will either fill up completely or will be only partially filled.

Chapter 15

Multi-threaded applications and asynchronous I/O

libusb is a thread-safe library, but extra considerations must be applied to applications which interact with libusb from multiple threads.

The underlying issue that must be addressed is that all libusb I/O revolves around monitoring file descriptors through the poll()/select() system calls. This is directly exposed at the asynchronous interface but it is important to note that the synchronous interface is implemented on top of the asynchonrous interface, therefore the same considerations apply.

The issue is that if two or more threads are concurrently calling poll() or select() on libusb's file descriptors then only one of those threads will be woken up when an event arrives. The others will be completely oblivious that anything has happened.

Consider the following pseudo-code, which submits an asynchronous transfer then waits for its completion. This style is one way you could implement a synchronous interface on top of the asynchronous interface (and libusb does something similar, albeit more advanced due to the complications explained on this page).

```
void cb(struct libusb_transfer *transfer)
{
        int *completed = transfer->user_data;
         completed = 1;
}

void myfunc() {
        struct libusb_transfer *transfer;
        unsigned char buffer[LIBUSB_CONTROL_SETUP_SIZE] __attribute__ ((
      aligned (2)));
        int completed = 0;

        transfer = libusb_alloc_transfer(0);
        libusb_fill_control_setup(buffer,
                LIBUSB_REQUEST_TYPE_VENDOR |
      LIBUSB_ENDPOINT_OUT, 0x04, 0x01, 0, 0);
        libusb_fill_control_transfer(transfer, dev, buffer, cb, &completed, 1000);
        libusb_submit_transfer(transfer);

        while (!completed) {
                poll(libusb file descriptors, 120*1000);
                if (poll indicates activity)
                        libusb_handle_events_timeout(ctx, &zero_tv);
        }
        printf("completed!");
        // other code here
}
```

Here we are *serializing* completion of an asynchronous event against a condition - the condition being completion of a specific transfer. The poll() loop has a long timeout to minimize CPU usage during situations when nothing is happening (it could reasonably be unlimited).

If this is the only thread that is polling libusb's file descriptors, there is no problem: there is no danger that another thread will swallow up the event that we are interested in. On the other hand, if there is another thread polling the same descriptors, there is a chance that it will receive the event that we were interested in. In this situation, `myfunc()` will only realise that the transfer has completed on the next iteration of the loop, *up to 120 seconds later.* Clearly a two-minute delay is undesirable, and don't even think about using short timeouts to circumvent this issue!

The solution here is to ensure that no two threads are ever polling the file descriptors at the same time. A naive implementation of this would impact the capabilities of the library, so libusb offers the scheme documented below to ensure no loss of functionality.

Before we go any further, it is worth mentioning that all libusb-wrapped event handling procedures fully adhere to the scheme documented below. This includes libusb_handle_events() and its variants, and all the synchronous I/O functions - libusb hides this headache from you.

15.1 libusb_handle_events() from multiple threads

Even when only using libusb_handle_events() and synchronous I/O functions, you can still have a race condition. You might be tempted to solve the above with libusb_handle_events() like so:

```
libusb_submit_transfer(transfer);
```

```
while (!completed) {
        libusb_handle_events(ctx);
}
printf("completed!");
```

This however has a race between the checking of completed and libusb_handle_events() acquiring the events lock, so another thread could have completed the transfer, resulting in this thread hanging until either a timeout or another event occurs. See also commit 6696512aade99bb15d6792af90ae329af270eba6 which fixes this in the synchronous API implementation of libusb.

Fixing this race requires checking the variable completed only after taking the event lock, which defeats the concept of just calling libusb_handle_events() without worrying about locking. This is why libusb-1.0.9 introduces the new libusb_handle_events_timeout_completed() and libusb_handle_events_completed() functions, which handles doing the completion check for you after they have acquired the lock:

```
libusb_submit_transfer(transfer);

while (!completed) {
        libusb_handle_events_completed(ctx, &completed);
}
printf("completed!");
```

This nicely fixes the race in our example. Note that if all you want to do is submit a single transfer and wait for its completion, then using one of the synchronous I/O functions is much easier.

15.2 The events lock

The problem is when we consider the fact that libusb exposes file descriptors to allow for you to integrate asynchronous USB I/O into existing main loops, effectively allowing you to do some work behind libusb's back. If you do take libusb's file descriptors and pass them to poll()/select() yourself, you need to be aware of the associated issues.

The first concept to be introduced is the events lock. The events lock is used to serialize threads that want to handle events, such that only one thread is handling events at any one time.

You must take the events lock before polling libusb file descriptors, using libusb_lock_events(). You must release the lock as soon as you have aborted your poll()/select() loop, using libusb_unlock_events().

15.3 Letting other threads do the work for you

Although the events lock is a critical part of the solution, it is not enough on it's own. You might wonder if the following is sufficient...

```
libusb_lock_events(ctx);
while (!completed) {
        poll(libusb file descriptors, 120*1000);
        if (poll indicates activity)
                libusb_handle_events_timeout(ctx, &zero_tv);
}
libusb_unlock_events(ctx);
```

...and the answer is that it is not. This is because the transfer in the code shown above may take a long time (say 30 seconds) to complete, and the lock is not released until the transfer is completed.

Another thread with similar code that wants to do event handling may be working with a transfer that completes after a few milliseconds. Despite having such a quick completion time, the other thread cannot check that status of its transfer until the code above has finished (30 seconds later) due to contention on the lock.

To solve this, libusb offers you a mechanism to determine when another thread is handling events. It also offers a mechanism to block your thread until the event handling thread has completed an event (and this mechanism does not involve polling of file descriptors).

After determining that another thread is currently handling events, you obtain the *event waiters* lock using libusb_lock_event_waiters(). You then re-check that some other thread is still handling events, and if so, you call libusb_wait_for_event().

libusb_wait_for_event() puts your application to sleep until an event occurs, or until a thread releases the events lock. When either of these things happen, your thread is woken up, and should re-check the condition it was waiting on. It should also re-check that another thread is handling events, and if not, it should start handling events itself.

This looks like the following, as pseudo-code:

```
retry:
if (libusb_try_lock_events(ctx) == 0) {
        // we obtained the event lock: do our own event handling
        while (!completed) {
                if (!libusb_event_handling_ok(ctx)) {
                        libusb_unlock_events(ctx);
                        goto retry;
                }
```

```
                poll(libusb file descriptors, 120*1000);
                if (poll indicates activity)
                        libusb_handle_events_locked(ctx, 0);
        }
        libusb_unlock_events(ctx);
} else {
        // another thread is doing event handling. wait for it to signal us that
        // an event has completed
        libusb_lock_event_waiters(ctx);

        while (!completed) {
                // now that we have the event waiters lock, double check that another
                // thread is still handling events for us. (it may have ceased handling
                // events in the time it took us to reach this point)
                if (!libusb_event_handler_active(ctx)) {
                        // whoever was handling events is no longer doing so, try again
                        libusb_unlock_event_waiters(ctx);
                        goto retry;
                }

                libusb_wait_for_event(ctx, NULL);
        }
        libusb_unlock_event_waiters(ctx);
}
printf("completed!\n");
```

A naive look at the above code may suggest that this can only support one event waiter (hence a total of 2 competing threads, the other doing event handling), because the event waiter seems to have taken the event waiters lock while waiting for an event. However, the system does support multiple event waiters, because libusb_wait_for_event() actually drops the lock while waiting, and reaquires it before continuing.

We have now implemented code which can dynamically handle situations where nobody is handling events (so we should do it ourselves), and it can also handle situations where another thread is doing event handling (so we can piggyback onto them). It is also equipped to handle a combination of the two, for example, another thread is doing event handling, but for whatever reason it stops doing so before our condition is met, so we take over the event handling.

Four functions were introduced in the above pseudo-code. Their importance should be apparent from the code shown above.

1. libusb_try_lock_events() is a non-blocking function which attempts to acquire the events lock but returns a failure code if it is contended.
2. libusb_event_handling_ok() checks that libusb is still happy for your thread to be performing event handling. Sometimes, libusb needs to interrupt the event handler, and this is how you can check if you have been interrupted. If this function returns 0, the correct behaviour is for you to give up the event handling lock, and then to repeat the cycle. The following libusb_try_lock_events() will fail, so you will become an events waiter. For more information on this, read The full story below.
3. libusb_handle_events_locked() is a variant of libusb_handle_events_timeout() that you can call while holding the events lock. libusb_handle_events_timeout() itself implements similar logic to the above, so be sure not to call it when you are "working behind libusb's back", as is the case here.
4. libusb_event_handler_active() determines if someone is currently holding the events lock

You might be wondering why there is no function to wake up all threads blocked on libusb_wait_for_event(). This is because libusb can do this internally: it will wake up all such threads when someone calls libusb_unlock_events() or when a transfer completes (at the point after its callback has returned).

15.3.1 The full story

The above explanation should be enough to get you going, but if you're really thinking through the issues then you may be left with some more questions regarding libusb's internals. If you're curious, read on, and if not, skip to the next section to avoid confusing yourself!

The immediate question that may spring to mind is: what if one thread modifies the set of file descriptors that need to be polled while another thread is doing event handling?

There are 2 situations in which this may happen.

1. libusb_open() will add another file descriptor to the poll set, therefore it is desirable to interrupt the event handler so that it restarts, picking up the new descriptor.
2. libusb_close() will remove a file descriptor from the poll set. There are all kinds of race conditions that could arise here, so it is important that nobody is doing event handling at this time.

libusb handles these issues internally, so application developers do not have to stop their event handlers while opening/closing devices. Here's how it works, focusing on the libusb_close() situation first:

1. During initialization, libusb opens an internal pipe, and it adds the read end of this pipe to the set of file descriptors to be polled.
2. During libusb_close(), libusb writes some dummy data on this event pipe. This immediately interrupts the event handler. libusb also records internally that it is trying to interrupt event handlers for this high-priority event.

3. At this point, some of the functions described above start behaving differently:
   - libusb_event_handling_ok() starts returning 1, indicating that it is NOT OK for event handling to continue.
   - libusb_try_lock_events() starts returning 1, indicating that another thread holds the event handling lock, even if the lock is uncontended.
   - libusb_event_handler_active() starts returning 1, indicating that another thread is doing event handling, even if that is not true.
4. The above changes in behaviour result in the event handler stopping and giving up the events lock very quickly, giving the high-priority libusb_close() operation a "free ride" to acquire the events lock. All threads that are competing to do event handling become event waiters.
5. With the events lock held inside libusb_close(), libusb can safely remove a file descriptor from the poll set, in the safety of knowledge that nobody is polling those descriptors or trying to access the poll set.
6. After obtaining the events lock, the close operation completes very quickly (usually a matter of milliseconds) and then immediately releases the events lock.
7. At the same time, the behaviour of libusb_event_handling_ok() and friends reverts to the original, documented behaviour.
8. The release of the events lock causes the threads that are waiting for events to be woken up and to start competing to become event handlers again. One of them will succeed; it will then re-obtain the list of poll descriptors, and USB I/O will then continue as normal.

libusb_open() is similar, and is actually a more simplistic case. Upon a call to libusb_open():

1. The device is opened and a file descriptor is added to the poll set.
2. libusb sends some dummy data on the event pipe, and records that it is trying to modify the poll descriptor set.
3. The event handler is interrupted, and the same behaviour change as for libusb_close() takes effect, causing all event handling threads to become event waiters.
4. The libusb_open() implementation takes its free ride to the events lock.
5. Happy that it has successfully paused the events handler, libusb_open() releases the events lock.
6. The event waiter threads are all woken up and compete to become event handlers again. The one that succeeds will obtain the list of poll descriptors again, which will include the addition of the new device.

15.3.2 Closing remarks

The above may seem a little complicated, but hopefully I have made it clear why such complications are necessary. Also, do not forget that this only applies to applications that take libusb's file descriptors and integrate them into their own polling loops.

You may decide that it is OK for your multi-threaded application to ignore some of the rules and locks detailed above, because you don't think that two threads can ever be polling the descriptors at the same time. If that is the case, then that's good news for you because you don't have to worry. But be careful here; remember that the synchronous I/O functions do event handling internally. If you have one thread doing event handling in a loop (without implementing the rules and locking semantics documented above) and another trying to send a synchronous USB transfer, you will end up with two threads monitoring the same descriptors, and the above-described undesirable behaviour occurring. The solution is for your polling thread to play by the rules; the synchronous I/O functions do so, and this will result in them getting along in perfect harmony.

If you do have a dedicated thread doing event handling, it is perfectly legal for it to take the event handling lock for long periods of time. Any synchronous I/O functions you call from other threads will transparently fall back to the "event waiters" mechanism detailed above. The only consideration that your event handling thread must apply is the one related to libusb_event_handling_ok(): you must call this before every poll(), and give up the events lock if instructed.

Chapter 16

libusb

![Build Status](https://travis-ci.org/libusb/libusb.svg?branch=master) ![Build status](https://ci.appveyor.com/api/projects/status/xvrfam94jii4a6lw?svg=true) ![Coverity Scan Build Status](https://scan.coverity.com/projects/2180/badge.-svg)

libusb is a library for USB device access from Linux, Mac OS X, Windows, OpenBSD/NetBSD and Haiku userspace. It is written in C (Haiku backend in C++) and licensed under the GNU Lesser General Public License version 2.1 or, at your option, any later version (see COPYING).

libusb is abstracted internally in such a way that it can hopefully be ported to other operating systems. Please see the PORTING file for more information.

libusb homepage: http://libusb.info/

Developers will wish to consult the API documentation: http://api.libusb.info

Use the mailing list for questions, comments, etc: http://mailing-list.libusb.info

- Pete Batard pete@akeo.ie
- Hans de Goede hdegoede@redhat.com
- Xiaofan Chen xiaofanc@gmail.com
- Ludovic Rousseau ludovic.rousseau@gmail.com
- Nathan Hjelm hjelmn@users.sourceforge.net
- Chris Dickens christopher.a.dickens@gmail.com

(Please use the mailing list rather than mailing developers directly)

Chapter 17

README

Chapter 18

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 19

How to run

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

Chapter 20

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 21

Native Utils

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

Notes

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

Usage

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

```
import cz.adamh.NativeUtils;

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

    public native void hello();
}
```

More information

More information can be found in accompanying `blog post`.

Chapter 22

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 23

How to run

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

Chapter 24

Todo List

Global AIOBufIteratorGetValue (AIOBufIterator *biter)

make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

Global AIOChannelMaskToString (AIOChannelMask *mask)

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

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

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

Global AIOContinuousBufReset (AIOContinuousBuf *buf)

Fix this to use condition variable

Fix this to use condition variable

Global AIODeviceTablePopulateTable (void)

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

Note

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

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;'

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

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

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

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

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

Page extcal.c

Setup BUILDING Tag

Page idio_sample.c

Complete this example

Page idio_sample2.c

Complete this example

Page iiro_sample.c

Complete this example

Global NewAIOChannelMaskFromStr (const char *bitfields)

Add smarter error checking

Add smarter error checking

Page Wrappers

Complete Wrapper Doxygen page

Chapter 25

Deprecated List

Page bulk_aquire_test.c

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

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

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

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

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Chapter 26

Module Index

26.1 Modules

Here is a list of all modules:

Library initialization/deinitialization 89
Device handling and enumeration 92
Miscellaneous 105
USB descriptors 110
Device hotplug event notification 118
Asynchronous device I/O 121
Polling and timing 131
Synchronous device I/O 140

Chapter 27

Namespace Index

27.1 Namespace List

Here is a list of all namespaces with brief descriptions:

AIOUSB 143

Chapter 28

Hierarchical Index

28.1 Class Hierarchy

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

ad_gain_pairs . . . 145
ADCConfigBlock . . . 145
ADRange . . . 146
AI16_DataPoint . . . 146
AI16_DataSet . . . 149
aio_callback_args . . . 152
aio_channel_range . . . 152
aio_counts_converter . . . 153
aio_either_val . . . 154
aio_ret_value . . . 154
AIOArgument . . . 154
AIOArguments . . . 155
AIOBuf . . . 155
aiobuf_iterator . . . 156
AIOChannelMask . . . 156
AIOChannelRangeTmp . . . 157
AIOCmd . . . 157
AIOCommandLineOptions . . . 157
AIOContinuousBuf . . . 159
AIODeviceInfo . . . 161
AIODeviceQuery . . . 161
aioerror . . . 162
AIOFifo . . . 163
AIOGainRange . . . 163
AIOProductGroup . . . 163
AIOProductRange . . . 164
aiousb_libusb_args . . . 164
AIOUSBDevice . . . 164
aiousboption . . . 168
AIOWDGConfig . . . 168
AnalogIORange . . . 188
- AI16_InputRange . . . 151
- AO16_OutputRange . . . 195
- DA12_OutputRange . . . 212

BLooper . . . 196
- RosterLooper . . . 267

BulkAcquireWorkerParams . . . 197
channel_range . . . 197
cJSON . . . 197
cJSON_Hooks . . . 198
command_block_wrapper . . . 198
command_status_wrapper . . . 199
config_options . . . 199
configuration . . . 199
Counter . . . 200
CStringArray . . . 207
darwin_cached_device . . . 213
darwin_device_handle_priv . . . 214
darwin_device_priv . . . 214
darwin_device_handle_priv::darwin_interface . . . 215
darwin_transfer_priv . . . 215

device_priv . . . 216
DeviceInfo . . . 216
DeviceProperties . . . 217
DeviceSubsystem . . . 218
  AnalogInputSubsystem . . . 168
  AnalogOutputSubsystem . . . 190
    AO16_AnalogOutputSubsystem . . . 192
    DA12_AnalogOutputSubsystem . . . 207
  CounterSubsystem . . . 204
  DigitalIOSubsystem . . . 219
  DIOStreamSubsystem . . . 225
devlink_cbarg . . . 219
DIOBuf . . . 224
discovered_devs . . . 229
driver_lookup . . . 229
endpoint . . . 230
exception
  Error . . . 230
fx_known_device . . . 230
handle_priv . . . 231
hid_device_priv . . . 231
HIDD_ATTRIBUTES . . . 232
HIDP_CAPS . . . 232
htab_entry . . . 233
imaxdiv_t . . . 234
interface_handle_t . . . 234
invalid_argument
  IllegalArgumentException . . . 233
libusb_bos_descriptor . . . 235
libusb_bos_dev_capability_descriptor . . . 236
libusb_config_descriptor . . . 236
libusb_container_id_descriptor . . . 238
libusb_context . . . 239
libusb_control_setup . . . 240
libusb_device . . . 241
libusb_device_descriptor . . . 242
libusb_device_handle . . . 244
libusb_endpoint_descriptor . . . 244
libusb_hid_descriptor . . . 246
libusb_hotplug_callback . . . 246
libusb_hotplug_message . . . 248
libusb_interface . . . 248
libusb_interface_descriptor . . . 249
libusb_iso_packet_descriptor . . . 250
libusb_pollfd . . . 251
libusb_ss_endpoint_companion_descriptor . . . 251
libusb_ss_usb_device_capability_descriptor . . . 252
libusb_testlib_ctx . . . 254
libusb_testlib_test . . . 254
libusb_transfer . . . 255
libusb_usb_2_0_extension_descriptor . . . 256
libusb_version . . . 257
linux_device_handle_priv . . . 258
linux_device_priv . . . 259
linux_transfer_priv . . . 259
list_head . . . 259
lookup . . . 260
mux_settings . . . 260
new_aio_fifo . . . 260
node_args . . . 261
option . . . 262
options . . . 262
opts . . . 263
OutputVoltagePoint . . . 265
pollfd . . . 266
ProductIDName . . . 266
ram_poke_context . . . 266
rangelookup . . . 267
runtime_error
  OperationFailedException . . . 261
sunos_device_handle_priv . . . 268
sunos_device_priv . . . 268

sunos_transfer_priv . . . 269
tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST . . . 269
tag_USB_DK_DEVICE_ID . . . 269
tag_USB_DK_DEVICE_INFO . . . 270
tag_USB_DK_GEN_TRANSFER_RESULT . . . 270
tag_USB_DK_ISO_TRANSFER_RESULT . . . 270
tag_USB_DK_TRANSFER_REQUEST . . . 271
tag_USB_DK_TRANSFER_RESULT . . . 271
TestCaseSetup . . . 272
timer_request . . . 276
timespec . . . 276
UKW_CONTROL_HEADER . . . 276
UKW_DEVICE_DESCRIPTOR . . . 277
USB_CONFIGURATION_DESCRIPTOR . . . 286
USB_CONFIGURATION_DESCRIPTOR_SHORT . . . 286
usb_descriptor_header . . . 294
USB_DESCRIPTOR_REQUEST . . . 295
USB_ENDPOINT_DESCRIPTOR . . . 302
USB_HUB_CAP_FLAGS . . . 303
USB_HUB_CAPABILITIES . . . 303
USB_HUB_CAPABILITIES_EX . . . 303
USB_HUB_DESCRIPTOR . . . 304
USB_HUB_INFORMATION . . . 304
USB_HUB_NAME_FIXED . . . 305
USB_INTERFACE_DESCRIPTOR . . . 305
USB_MI_PARENT_INFORMATION . . . 306
USB_NODE_CONNECTION_INFORMATION_EX . . . 306
USB_NODE_CONNECTION_NAME . . . 306
USB_NODE_CONNECTION_NAME_FIXED . . . 307
USB_NODE_INFORMATION . . . 307
USB_PIPE_INFO . . . 307
usb_raw_command . . . 308
USB_ROOT_HUB_NAME . . . 311
USB_ROOT_HUB_NAME_FIXED . . . 311
USBDevice . . . 311
USBDeviceBase . . . 314
- USB_AI16_Family . . . 278
- USB_AIO16_Family . . . 280
- USB_AO16_Family . . . 283
- USB_CTR_15_Family . . . 287
- USB_DA12_8A_Family . . . 289
- USB_DA12_8E_Family . . . 291
- USB_DIO_16_Family . . . 295
- USB_DIO_32_Family . . . 298
- USB_DIO_Family . . . 300

USBDeviceHandle . . . 318
USBDeviceManager . . . 320
usbfs_bulktransfer . . . 326
usbfs_connectinfo . . . 326
usbfs_ctrltransfer . . . 326
usbfs_disconnect_claim . . . 327
usbfs_getdriver . . . 327
usbfs_hub_portinfo . . . 328
usbfs_ioctl . . . 328
usbfs_iso_packet_desc . . . 328
usbfs_setinterface . . . 329
usbfs_streams . . . 329
usbfs_urb . . . 329
usbi_cond . . . 330
usbi_cond_perthread . . . 330
usbi_os_backend . . . 331
usbi_pollfd . . . 333
usbi_transfer . . . 333
USBRoster . . . 334
USBTransfer . . . 334
ushort_array . . . 335
vector
- AI16_DataPointArray . . . 148
- BoolArray . . . 196
- CounterList . . . 204
- DoubleArray . . . 229
- IntArray . . . 234

OutputVoltagePointArray . . . 265
StringArray . . . 268
UCharArray . . . 276
USBDeviceArray . . . 314
UShortArray . . . 335
WatchedEntry . . . 336
wince_device_priv . . . 336
wince_transfer_priv . . . 337
windows_device_handle_priv . . . 337
windows_device_priv . . . 337
windows_transfer_priv . . . 338
windows_usb_api_backend . . . 339
winfd . . . 340
winusb_interface . . . 340
WINUSB_PIPE_INFORMATION . . . 342
WINUSB_SETUP_PACKET . . . 342

Chapter 29

Data Structure Index

29.1 Data Structures

Here are the data structures with brief descriptions:

ad_gain_pairs . . . 145
ADCConfigBlock . . . 145
ADRange . . . 146
AI16_DataPoint
Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device . . . 146
AI16_DataPointArray . . . 148
AI16_DataSet
Class AI16_DataSet represents a data set captured from a USB_AI16_Family device . . . 149
AI16_InputRange . . . 151
aio_callback_args . . . 152
aio_channel_range . . . 152
aio_counts_converter . . . 153
aio_either_val . . . 154
aio_ret_value . . . 154
AIOArgument . . . 154
AIOArguments . . . 155
AIOBuf . . . 155
aiobuf_iterator . . . 156
AIOChannelMask . . . 156
AIOChannelRangeTmp . . . 157
AIOCmd . . . 157
AIOCommandLineOptions . . . 157
AIOContinuousBuf
AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API . 159
AIODeviceInfo . . . 161
AIODeviceQuery . . . 161
aioerror . . . 162
AIOFifo
AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition . . . 163
AIOGainRange . . . 163
AIOProductGroup
A smart product group that marks a range of ACCES I/O Products . . . 163
AIOProductRange
A simplified range of Products based off of device ids . . . 164
aiousb_libusb_args . . . 164
AIOUSBDevice . . . 164
aiousboption . . . 168
AIOWDGConfig . . . 168
AnalogInputSubsystem
Class AnalogInputSubsystem represents the analog input subsystem of a device . . . 168
AnalogIORange
Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities . . . 188
AnalogOutputSubsystem
Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device . . . 190
AO16_AnalogOutputSubsystem
Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device . . . 192
AO16_OutputRange . . . 195
BLooper . . . 196
BoolArray . . . 196
BulkAcquireWorkerParams . . . 197

channel_range . . . 197
cJSON . . . 197
cJSON_Hooks . . . 198
command_block_wrapper . . . 198
command_status_wrapper . . . 199
config_options . . . 199
configuration . . . 199
Counter
Class Counter represents a single counter/timer . . . 200
CounterList . . . 204
CounterSubsystem
Class CounterSubsystem represents the counter/timer subsystem of a device . . . 204
CStringArray . . . 207
DA12_AnalogOutputSubsystem
Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device . . . 207
DA12_OutputRange . . . 212
darwin_cached_device . . . 213
darwin_device_handle_priv . . . 214
darwin_device_priv . . . 214
darwin_device_handle_priv::darwin_interface . . . 215
darwin_transfer_priv . . . 215
device_priv . . . 216
DeviceInfo . . . 216
DeviceProperties
Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management . . . 217
DeviceSubsystem
Class DeviceSubsystem is the abstract super class for all device subsystems . . . 218
devlink_cbarg . . . 219
DigitalIOSubsystem
Class DigitalIOSubsystem represents the digital I/O subsystem of a device . . . 219
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 . . . 224
DIOStreamSubsystem
Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device . . . 225
discovered_devs . . . 229
DoubleArray . . . 229
driver_lookup . . . 229
endpoint . . . 230
Error . . . 230
fx_known_device . . . 230
handle_priv . . . 231
hid_device_priv . . . 231
HIDD_ATTRIBUTES . . . 232
HIDP_CAPS . . . 232
htab_entry . . . 233
IllegalArgumentException
Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method . . . 233
imaxdiv_t
This file has no copyright assigned and is placed in the Public Domain . . . 234
IntArray . . . 234
interface_handle_t . . . 234
libusb_bos_descriptor
A structure representing the Binary Device Object Store (BOS) descriptor . . . 235
libusb_bos_dev_capability_descriptor
A generic representation of a BOS Device Capability descriptor . . . 236
libusb_config_descriptor
A structure representing the standard USB configuration descriptor . . . 236
libusb_container_id_descriptor
A structure representing the Container ID descriptor . . . 238
libusb_context . . . 239
libusb_control_setup
Setup packet for control transfers . . . 240
libusb_device . . . 241
libusb_device_descriptor
A structure representing the standard USB device descriptor . . . 242
libusb_device_handle . . . 244
libusb_endpoint_descriptor
A structure representing the standard USB endpoint descriptor . . . 244
libusb_hid_descriptor . . . 246

libusb_hotplug_callback
The hotplug callback structure . . . 246
libusb_hotplug_message . . . 248
libusb_interface
A collection of alternate settings for a particular USB interface . . . 248
libusb_interface_descriptor
A structure representing the standard USB interface descriptor . . . 249
libusb_iso_packet_descriptor
Isochronous packet descriptor . . . 250
libusb_pollfd
File descriptor for polling . . . 251
libusb_ss_endpoint_companion_descriptor
A structure representing the superspeed endpoint companion descriptor . . . 251
libusb_ss_usb_device_capability_descriptor
A structure representing the SuperSpeed USB Device Capability descriptor This descriptor is documented in section 9.6.2.2 of the USB 3.0 specification . . . 252
libusb_testlib_ctx
Context for test library functions . . . 254
libusb_testlib_test
Structure holding a test description . . . 254
libusb_transfer
The generic USB transfer structure . . . 255
libusb_usb_2_0_extension_descriptor
A structure representing the USB 2.0 Extension descriptor This descriptor is documented in section 9.6.2.1 of the USB 3.0 specification . . . 256
libusb_version
Structure providing the version of the libusb runtime . . . 257
linux_device_handle_priv . . . 258
linux_device_priv . . . 259
linux_transfer_priv . . . 259
list_head . . . 259
lookup . . . 260
mux_settings . . . 260
new_aio_fifo . . . 260
node_args . . . 261
OperationFailedException
Class OperationFailedException is thrown whenever an operation attempted on a device fails . . . 261
option . . . 262
options . . . 262
opts . . . 263
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 . . . 265
OutputVoltagePointArray . . . 265
pollfd . . . 266
ProductIDName . . . 266
ram_poke_context . . . 266
rangelookup . . . 267
RosterLooper . . . 267
StringArray . . . 268
sunos_device_handle_priv . . . 268
sunos_device_priv . . . 268
sunos_transfer_priv . . . 269
tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST . . . 269
tag_USB_DK_DEVICE_ID . . . 269
tag_USB_DK_DEVICE_INFO . . . 270
tag_USB_DK_GEN_TRANSFER_RESULT . . . 270
tag_USB_DK_ISO_TRANSFER_RESULT . . . 270
tag_USB_DK_TRANSFER_REQUEST . . . 271
tag_USB_DK_TRANSFER_RESULT . . . 271
TestCaseSetup . . . 272
timer_request . . . 276
timespec . . . 276
UCharArray . . . 276
UKW_CONTROL_HEADER . . . 276
UKW_DEVICE_DESCRIPTOR . . . 277
USB_AI16_Family . . . 278

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 . . . 280
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 . . . 283
USB_CONFIGURATION_DESCRIPTOR . . . 286
USB_CONFIGURATION_DESCRIPTOR_SHORT . . . 286
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 . . . 287
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 . . . 289
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 . . . 291
usb_descriptor_header . . . 294
USB_DESCRIPTOR_REQUEST . . . 295
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 . . . 295
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 . . . 298
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 . . . 300
USB_ENDPOINT_DESCRIPTOR . . . 302
USB_HUB_CAP_FLAGS . . . 303
USB_HUB_CAPABILITIES . . . 303
USB_HUB_CAPABILITIES_EX . . . 303
USB_HUB_DESCRIPTOR . . . 304
USB_HUB_INFORMATION . . . 304
USB_HUB_NAME_FIXED . . . 305
USB_INTERFACE_DESCRIPTOR . . . 305
USB_MI_PARENT_INFORMATION . . . 306
USB_NODE_CONNECTION_INFORMATION_EX . . . 306
USB_NODE_CONNECTION_NAME . . . 306
USB_NODE_CONNECTION_NAME_FIXED . . . 307
USB_NODE_INFORMATION . . . 307
USB_PIPE_INFO . . . 307
usb_raw_command . . . 308
USB_ROOT_HUB_NAME . . . 311
USB_ROOT_HUB_NAME_FIXED . . . 311
USBDevice . . . 311
USBDeviceArray . . . 314
USBDeviceBase
Class USBDeviceBase is the abstract super class of all USB device families . . . 314
USBDeviceHandle . . . 318
USBDeviceManager
Class USBDeviceManager manages all the USB devices on the bus . . . 320
usbfs_bulktransfer . . . 326
usbfs_connectinfo . . . 326
usbfs_ctrltransfer . . . 326
usbfs_disconnect_claim . . . 327
usbfs_getdriver . . . 327
usbfs_hub_portinfo . . . 328
usbfs_ioctl . . . 328
usbfs_iso_packet_desc . . . 328
usbfs_setinterface . . . 329

usbfs_streams . . . 329
usbfs_urb . . . 329
usbi_cond . . . 330
usbi_cond_perthread . . . 330
usbi_os_backend . . . 331
usbi_pollfd . . . 333
usbi_transfer . . . 333
USBRoster . . . 334
USBTransfer . . . 334
ushort_array . . . 335
UShortArray . . . 335
WatchedEntry . . . 336
wince_device_priv . . . 336
wince_transfer_priv . . . 337
windows_device_handle_priv . . . 337
windows_device_priv . . . 337
windows_transfer_priv . . . 338
windows_usb_api_backend . . . 339
winfd . . . 340
winusb_interface . . . 340
WINUSB_PIPE_INFORMATION . . . 342
WINUSB_SETUP_PACKET . . . 342

Chapter 30

File Index

30.1 File List

Here is a list of all files with brief descriptions:

deprecated/classlib/AI16_DataPoint.cpp . . . 343
deprecated/classlib/AI16_DataPoint.hpp . . . 343
deprecated/classlib/AI16_DataSet.cpp . . . 343
deprecated/classlib/AI16_DataSet.hpp . . . 343
deprecated/classlib/AI16_InputRange.cpp . . . 344
deprecated/classlib/AI16_InputRange.hpp . . . 344
deprecated/classlib/AnalogInputSubsystem.cpp
Class AnalogInputSubsystem implementation . . . 344
deprecated/classlib/AnalogInputSubsystem.hpp . . . 345
deprecated/classlib/AnalogIORange.cpp . . . 345
deprecated/classlib/AnalogIORange.hpp . . . 345
deprecated/classlib/AnalogOutputSubsystem.cpp . . . 345
deprecated/classlib/AnalogOutputSubsystem.hpp . . . 346
deprecated/classlib/AO16_AnalogOutputSubsystem.cpp . . . 346
deprecated/classlib/AO16_AnalogOutputSubsystem.hpp . . . 346
deprecated/classlib/AO16_OutputRange.cpp . . . 346
deprecated/classlib/AO16_OutputRange.hpp . . . 346
deprecated/classlib/Counter.cpp
Class Counter implementation . . . 347
deprecated/classlib/Counter.hpp . . . 347
deprecated/classlib/CounterSubsystem.cpp
Class CounterSubsystem implementation . . . 348
deprecated/classlib/CounterSubsystem.hpp . . . 348
deprecated/classlib/CppCommon.h . . . 348
deprecated/classlib/DA12_AnalogOutputSubsystem.cpp
Class DA12_AnalogOutputSubsystem implementation . . . 348
deprecated/classlib/DA12_AnalogOutputSubsystem.hpp . . . 349
deprecated/classlib/DA12_OutputRange.cpp . . . 349
deprecated/classlib/DA12_OutputRange.hpp . . . 349
deprecated/classlib/DeviceSubsystem.cpp . . . 350
deprecated/classlib/DeviceSubsystem.hpp . . . 350
deprecated/classlib/DigitalIOSubsystem.cpp
Class DigitalIOSubsystem implementation . . . 350
deprecated/classlib/DigitalIOSubsystem.hpp . . . 350
deprecated/classlib/DIOStreamSubsystem.cpp
Class DIOStreamSubsystem implementation . . . 351
deprecated/classlib/DIOStreamSubsystem.hpp . . . 351
deprecated/classlib/OutputVoltagePoint.hpp . . . 351
deprecated/classlib/USB_AI16_Family.cpp . . . 352
deprecated/classlib/USB_AI16_Family.hpp . . . 352
deprecated/classlib/USB_AIO16_Family.cpp . . . 353
deprecated/classlib/USB_AIO16_Family.hpp . . . 353
deprecated/classlib/USB_AO16_Family.cpp . . . 353
deprecated/classlib/USB_AO16_Family.hpp . . . 353
deprecated/classlib/USB_CTR_15_Family.cpp . . . 354
deprecated/classlib/USB_CTR_15_Family.hpp . . . 354
deprecated/classlib/USB_DA12_8A_Family.cpp . . . 354
deprecated/classlib/USB_DA12_8A_Family.hpp . . . 354
deprecated/classlib/USB_DA12_8E_Family.cpp . . . 355
deprecated/classlib/USB_DA12_8E_Family.hpp . . . 355
deprecated/classlib/USB_DIO_16_Family.cpp . . . 355
deprecated/classlib/USB_DIO_16_Family.hpp . . . 355

deprecated/classlib/USB_DIO_32_Family.cpp . . . 356
deprecated/classlib/USB_DIO_32_Family.hpp . . . 356
deprecated/classlib/USB_DIO_Family.cpp . . . 356
deprecated/classlib/USB_DIO_Family.hpp . . . 356
deprecated/classlib/USBDeviceBase.cpp . . . 357
deprecated/classlib/USBDeviceBase.hpp . . . 357
deprecated/classlib/USBDeviceManager.cpp
Class USBDeviceManager implementation . . . 358
deprecated/classlib/USBDeviceManager.hpp
Class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations . . . 358
lib/ADCConfigBlock.c . . . 359
lib/ADCConfigBlock.h . . . 366
lib/AIOBuf.c . . . 374
lib/AIOBuf.h . . . 376
lib/AIOChannelMask.c . . . 380
lib/AIOChannelMask.h . . . 384
lib/AIOChannelRange.c . . . 390
lib/AIOChannelRange.h . . . 392
lib/AIOCmd.c . . . 395
lib/AIOCmd.h . . . 395
lib/AIOCommandLine.c . . . 396
lib/AIOCommandLine.h . . . 402
lib/AIOConfiguration.c . . . 410
lib/AIOConfiguration.h . . . 411
lib/AIOContinuousBuffer.c . . . 414
lib/AIOContinuousBuffer.h . . . 435
lib/AIOCountsConverter.c . . . 451
lib/AIOCountsConverter.h . . . 453
lib/AIODeviceInfo.c . . . 456
lib/AIODeviceInfo.h . . . 457
lib/AIODeviceQuery.c . . . 458
lib/AIODeviceQuery.h . . . 464
lib/AIODeviceTable.c . . . 469
lib/AIODeviceTable.h . . . 477
lib/AIOEither.c . . . 484
lib/AIOEither.h . . . 486
lib/AIOFifo.c . . . 490
lib/AIOFifo.h . . . 494
lib/AIOList.c . . . 499
lib/AIOList.h . . . 501
lib/AIOProductTypes.c . . . 504
lib/AIOProductTypes.h . . . 506
lib/AIOTuple.c . . . 510
lib/AIOTuple.h . . . 510
lib/AIOTypes.h . . . 512
lib/aiousb.h . . . 556
lib/AIOUSB_ADC.c . . . 556
lib/AIOUSB_ADC.h . . . 585
lib/AIOUSB_Core.c . . . 609
lib/AIOUSB_Core.h . . . 615
lib/AIOUSB_CTR.c . . . 623
lib/AIOUSB_CTR.h . . . 627
lib/AIOUSB_CustomEEPROM.c . . . 629
lib/AIOUSB_CustomEEPROM.h . . . 631
lib/AIOUSB_DAC.c . . . 632
lib/AIOUSB_DAC.h . . . 635
lib/AIOUSB_DIO.c . . . 638
lib/AIOUSB_DIO.h . . . 642
lib/AIOUSB_Log.c . . . 646
lib/AIOUSB_Log.h . . . 647
lib/AIOUSB_Properties.c . . . 651
lib/AIOUSB_Properties.h . . . 654
lib/AIOUSB_USB.c . . . 658
lib/AIOUSB_USB.h . . . 658
lib/AIOUSB_WDG.c . . . 659
lib/AIOUSB_WDG.h . . . 662
lib/AIOUSBDevice.c . . . 665
lib/AIOUSBDevice.h . . . 667
lib/cJSON.c . . . 670
lib/cJSON.h . . . 674
lib/CStringArray.c . . . 679
lib/CStringArray.h . . . 680

lib/DIOBuf.c . . . 682
lib/DIOBuf.h . . . 687
lib/USBDevice.c . . . 749
lib/USBDevice.h . . . 752
lib/mocks/mock_aiocontbuf_get_data.c . . . 693
lib/mocks/mock_aiocontbuf_get_data_arduino.c . . . 694
lib/mocks/mock_capture_usb.c
This file will allow capturing of all USB traffic, in and out . . . 694
lib/mocks/mock_dio.c . . . 696
lib/mocks/mock_usb_xfers.c . . . 697
lib/ndkbuild/jni/ADCConfigBlock.c . . . 363
lib/ndkbuild/jni/ADCConfigBlock.h . . . 371
lib/ndkbuild/jni/AIOBuf.c . . . 375
lib/ndkbuild/jni/AIOBuf.h . . . 378
lib/ndkbuild/jni/AIOChannelMask.c . . . 382
lib/ndkbuild/jni/AIOChannelMask.h . . . 387
lib/ndkbuild/jni/AIOChannelRange.c . . . 391
lib/ndkbuild/jni/AIOChannelRange.h . . . 393
lib/ndkbuild/jni/AIOCmd.c . . . 395
lib/ndkbuild/jni/AIOCmd.h . . . 396
lib/ndkbuild/jni/AIOCommandLine.c . . . 399
lib/ndkbuild/jni/AIOCommandLine.h . . . 406
lib/ndkbuild/jni/AIOConfiguration.c . . . 410
lib/ndkbuild/jni/AIOConfiguration.h . . . 412
lib/ndkbuild/jni/AIOContinuousBuffer.c . . . 425
lib/ndkbuild/jni/AIOContinuousBuffer.h . . . 442
lib/ndkbuild/jni/AIOCountsConverter.c . . . 452
lib/ndkbuild/jni/AIOCountsConverter.h . . . 454
lib/ndkbuild/jni/AIODeviceInfo.c . . . 456
lib/ndkbuild/jni/AIODeviceInfo.h . . . 457
lib/ndkbuild/jni/AIODeviceQuery.c . . . 461
lib/ndkbuild/jni/AIODeviceQuery.h . . . 467
lib/ndkbuild/jni/AIODeviceTable.c . . . 473
lib/ndkbuild/jni/AIODeviceTable.h . . . 480
lib/ndkbuild/jni/AIOEither.c . . . 485
lib/ndkbuild/jni/AIOEither.h . . . 488
lib/ndkbuild/jni/AIOFifo.c . . . 492
lib/ndkbuild/jni/AIOFifo.h . . . 496
lib/ndkbuild/jni/AIOList.c . . . 500
lib/ndkbuild/jni/AIOList.h . . . 502
lib/ndkbuild/jni/AIOProductTypes.c . . . 505
lib/ndkbuild/jni/AIOProductTypes.h . . . 508
lib/ndkbuild/jni/AIOTuple.c . . . 510
lib/ndkbuild/jni/AIOTuple.h . . . 511
lib/ndkbuild/jni/AIOTypes.h . . . 534
lib/ndkbuild/jni/aiousb.h . . . 556
lib/ndkbuild/jni/AIOUSB_ADC.c . . . 570
lib/ndkbuild/jni/AIOUSB_ADC.h . . . 598
lib/ndkbuild/jni/AIOUSB_Core.c . . . 612
lib/ndkbuild/jni/AIOUSB_Core.h . . . 619
lib/ndkbuild/jni/AIOUSB_CTR.c . . . 625
lib/ndkbuild/jni/AIOUSB_CTR.h . . . 628
lib/ndkbuild/jni/AIOUSB_CustomEEPROM.c . . . 630
lib/ndkbuild/jni/AIOUSB_CustomEEPROM.h . . . 631
lib/ndkbuild/jni/AIOUSB_DAC.c . . . 633
lib/ndkbuild/jni/AIOUSB_DAC.h . . . 636
lib/ndkbuild/jni/AIOUSB_DIO.c . . . 640
lib/ndkbuild/jni/AIOUSB_DIO.h . . . 644
lib/ndkbuild/jni/AIOUSB_Log.c . . . 646
lib/ndkbuild/jni/AIOUSB_Log.h . . . 649
lib/ndkbuild/jni/AIOUSB_Properties.c . . . 652
lib/ndkbuild/jni/AIOUSB_Properties.h . . . 656
lib/ndkbuild/jni/AIOUSB_USB.c . . . 658
lib/ndkbuild/jni/AIOUSB_USB.h . . . 658
lib/ndkbuild/jni/AIOUSB_WDG.c . . . 661
lib/ndkbuild/jni/AIOUSB_WDG.h . . . 664
lib/ndkbuild/jni/AIOUSBDevice.c . . . 666
lib/ndkbuild/jni/AIOUSBDevice.h . . . 669
lib/ndkbuild/jni/cJSON.c . . . 672
lib/ndkbuild/jni/cJSON.h . . . 676
lib/ndkbuild/jni/CStringArray.c . . . 680
lib/ndkbuild/jni/CStringArray.h . . . 681

lib/ndkbuild/jni/DIOBuf.c . . . 684
lib/ndkbuild/jni/DIOBuf.h . . . 690
lib/ndkbuild/jni/libaiousb.h . . . 698
lib/ndkbuild/jni/USBDevice.c . . . 748
lib/ndkbuild/jni/USBDevice.h . . . 750
lib/ndkbuild/libusb/android/config.h . . . 754
lib/ndkbuild/libusb/examples/dpfp.c . . . 756
lib/ndkbuild/libusb/examples/dpfp_threaded.c . . . 757
lib/ndkbuild/libusb/examples/ezusb.c . . . 758
lib/ndkbuild/libusb/examples/ezusb.h . . . 759
lib/ndkbuild/libusb/examples/fxload.c . . . 761
lib/ndkbuild/libusb/examples/hotplugtest.c . . . 764
lib/ndkbuild/libusb/examples/listdevs.c . . . 764
lib/ndkbuild/libusb/examples/sam3u_benchmark.c . . . 764
lib/ndkbuild/libusb/examples/testlibusb.c . . . 765
lib/ndkbuild/libusb/examples/xusb.c . . . 765
lib/ndkbuild/libusb/examples/getopt/getopt.c . . . 761
lib/ndkbuild/libusb/examples/getopt/getopt.h . . . 762
lib/ndkbuild/libusb/examples/getopt/getopt1.c . . . 763
lib/ndkbuild/libusb/libusb/core.c . . . 767
lib/ndkbuild/libusb/libusb/descriptor.c . . . 769
lib/ndkbuild/libusb/libusb/hotplug.c . . . 771
lib/ndkbuild/libusb/libusb/hotplug.h . . . 771
lib/ndkbuild/libusb/libusb/io.c . . . 772
lib/ndkbuild/libusb/libusb/libusb.h . . . 773
lib/ndkbuild/libusb/libusb/libusbi.h . . . 781
lib/ndkbuild/libusb/libusb/strerror.c . . . 831
lib/ndkbuild/libusb/libusb/sync.c . . . 832
lib/ndkbuild/libusb/libusb/version.h . . . 832
lib/ndkbuild/libusb/libusb/version_nano.h . . . 832
lib/ndkbuild/libusb/libusb/os/darwin_usb.c . . . 786
lib/ndkbuild/libusb/libusb/os/darwin_usb.h . . . 786
lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp . . . 787
lib/ndkbuild/libusb/libusb/os/haiku_usb.h . . . 787
lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp . . . 788
lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.cpp . . . 788
lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.h . . . 788
lib/ndkbuild/libusb/libusb/os/linux_netlink.c . . . 790
lib/ndkbuild/libusb/libusb/os/linux_udev.c . . . 790
lib/ndkbuild/libusb/libusb/os/linux_usbfs.c . . . 791
lib/ndkbuild/libusb/libusb/os/linux_usbfs.h . . . 792
lib/ndkbuild/libusb/libusb/os/netbsd_usb.c . . . 795
lib/ndkbuild/libusb/libusb/os/openbsd_usb.c . . . 795
lib/ndkbuild/libusb/libusb/os/poll_posix.c . . . 796
lib/ndkbuild/libusb/libusb/os/poll_posix.h . . . 796
lib/ndkbuild/libusb/libusb/os/poll_windows.c . . . 797
lib/ndkbuild/libusb/libusb/os/poll_windows.h . . . 798
lib/ndkbuild/libusb/libusb/os/sunos_usb.c . . . 801
lib/ndkbuild/libusb/libusb/os/sunos_usb.h . . . 801
lib/ndkbuild/libusb/libusb/os/threads_posix.c . . . 802
lib/ndkbuild/libusb/libusb/os/threads_posix.h . . . 802
lib/ndkbuild/libusb/libusb/os/threads_windows.c . . . 803
lib/ndkbuild/libusb/libusb/os/threads_windows.h . . . 804
lib/ndkbuild/libusb/libusb/os/wince_usb.c . . . 806
lib/ndkbuild/libusb/libusb/os/wince_usb.h . . . 806
lib/ndkbuild/libusb/libusb/os/windows_common.h . . . 809
lib/ndkbuild/libusb/libusb/os/windows_nt_common.c . . . 811
lib/ndkbuild/libusb/libusb/os/windows_nt_common.h . . . 812
lib/ndkbuild/libusb/libusb/os/windows_usbdk.c . . . 813
lib/ndkbuild/libusb/libusb/os/windows_usbdk.h . . . 813
lib/ndkbuild/libusb/libusb/os/windows_winusb.c . . . 816
lib/ndkbuild/libusb/libusb/os/windows_winusb.h . . . 818
lib/ndkbuild/libusb/msvc/config.h . . . 755
lib/ndkbuild/libusb/msvc/errno.h . . . 833
lib/ndkbuild/libusb/msvc/inttypes.h . . . 834
lib/ndkbuild/libusb/msvc/missing.c . . . 841
lib/ndkbuild/libusb/msvc/missing.h . . . 841
lib/ndkbuild/libusb/msvc/stdint.h . . . 841
lib/ndkbuild/libusb/tests/libusb_testlib.h . . . 845
lib/ndkbuild/libusb/tests/stress.c . . . 847
lib/ndkbuild/libusb/tests/testlib.c . . . 847
lib/ndkbuild/libusb/Xcode/config.h . . . 755

lib/wrappers/scilab/foo.c . . . 848
lib/wrappers/scilab/foo.h . . . 848
samples/TestLib/aiocommon.c . . . 849
samples/TestLib/aiocommon.h . . . 850
samples/TestLib/TestCaseSetup.cpp . . . 850
samples/TestLib/TestCaseSetup.h . . . 851
samples/USB-AI16-16/bulk_acquire_sample.c . . . 852
samples/USB-AI16-16/burst_test.c . . . 853
samples/USB-AI16-16/continuous_mode.c . . . 854
samples/USB-AI16-16/continuous_mode_callback.c . . . 854
samples/USB-AI16-16/continuous_mode_from_json_config.c . . . 855
samples/USB-AI16-16/daitest.cpp . . . 856
samples/USB-AI16-16/dio_sample.c . . . 856
samples/USB-AI16-16/diotest.c . . . 857
samples/USB-AI16-16/diotest2.cpp . . . 857
samples/USB-AI16-16/read_channels_test.c . . . 862
samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp . . . 862
samples/USB-AI16-16/reverse_cal_table.cpp . . . 858
samples/USB-AI16-16/sample.cpp . . . 863
samples/USB-AI16-16/simp_test.cpp . . . 864
samples/USB-AI16-16/simple_continuous_with_json.c
Sample that demonstrates data ac . . . 864
samples/USB-AI16-16/start_stop_continuous.c . . . 865
samples/USB-AI16-16/test_fastscan.cpp . . . 865
samples/USB-AI16-16/HOLD/dirktest.c . . . 857
samples/USB-AI16-16/HOLD/julian_test.c . . . 858
samples/USB-AI16-16/HOLD/reverse_cal_table.cpp . . . 858
samples/USB-AI16-16/HOLD/sample_dio.c . . . 859
samples/USB-AI16-16/HOLD/slow_receiver_test.cpp . . . 859
samples/USB-AI16-16/HOLD/test.c . . . 860
samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp . . . 860
samples/USB-AI16-16/jni/read_channels_test.c . . . 861
samples/USB-AO16-16/sample.cpp . . . 863
samples/USB-ARB1/stream_test.c . . . 866
samples/USB-DA12-8A/sample.cpp . . . 863
samples/USB-DA12-8A/SampleClass.cpp . . . 866
samples/USB-DIO-16/receiver.cpp . . . 866
samples/USB-DIO-16/sample.cpp . . . 863
samples/USB-DIO-16/standalone_receiver.c . . . 867
samples/USB-DIO-32/daisample.c . . . 867
samples/USB-DIO-32/read_and_write_sample.c . . . 867
samples/USB-DIO-32/sample3.c . . . 869
samples/USB-DIO-48/read_and_write.c
AIOUSB sample program that writes and reads from a USB-DIO-48 . . . 870
samples/USB-DIO-96/dio96_read_write.c . . . 871
samples/USB-DIO-96/mytest.c . . . 872
samples/USB-DIO-96/read_and_write_sample.c . . . 868
samples/USB-DIO-96/tmp.c . . . 874
samples/USB-DIO-96/write_sample.c . . . 875
samples/USB-IDIO-16_8/idio_sample.c . . . 877
samples/USB-IDIO-16_8/idio_sample2.c . . . 878
samples/USB-IDIO-16_8/perftest.c . . . 879
samples/USB-IIRO-16_8/iiro_sample.c . . . 879

Chapter 31

Module Documentation

31.1 Library initialization/deinitialization

This page details how to initialize and deinitialize libusb.

Data Structures

- struct libusb_version

  Structure providing the version of the libusb runtime.

Typedefs

- typedef struct libusb_context libusb_context

  Structure representing a libusb session.

Enumerations

- enum libusb_log_level {
  LIBUSB_LOG_LEVEL_NONE = 0, LIBUSB_LOG_LEVEL_ERROR, LIBUSB_LOG_LEVEL_WARNING, LIBUSB_LOG_LEVEL_INFO,
  LIBUSB_LOG_LEVEL_DEBUG }

  Log message levels.

Functions

- void API_EXPORTED libusb_set_debug (libusb_context *ctx, int level)

  Set log message verbosity.
- int API_EXPORTED libusb_init (libusb_context **context)

  Initialize libusb.
- void API_EXPORTED libusb_exit (struct libusb_context *ctx)

  Deinitialize libusb.

31.1.1 Detailed Description

This page details how to initialize and deinitialize libusb. Initialization must be performed before using any libusb functionality, and similarly you must not call any libusb functions after deinitialization.

31.1.2 Typedef Documentation

typedef struct libusb_context libusb_context

Structure representing a libusb session.

The concept of individual libusb sessions allows for your program to use two libraries (or dynamically load two modules) which both independently use libusb. This will prevent interference between the individual libusb users - for example libusb_set_debug() will not affect the other user of the library, and libusb_exit() will not destroy resources that the other user is still using.

Sessions are created by libusb_init() and destroyed through libusb_exit(). If your application is guaranteed to only ever include a single libusb user (i.e. you), you do not have to worry about contexts: pass NULL in every function call where a context is required. The default context will be used.

For more information, see Contexts.

31.1.3 Enumeration Type Documentation

enum libusb_log_level

Log message levels.

- LIBUSB_LOG_LEVEL_NONE (0) : no messages ever printed by the library (default)
- LIBUSB_LOG_LEVEL_ERROR (1) : error messages are printed to stderr
- LIBUSB_LOG_LEVEL_WARNING (2) : warning and error messages are printed to stderr
- LIBUSB_LOG_LEVEL_INFO (3) : informational messages are printed to stdout, warning and error messages are printed to stderr
- LIBUSB_LOG_LEVEL_DEBUG (4) : debug and informational messages are printed to stdout, warnings and errors to stderr

Enumerator

LIBUSB_LOG_LEVEL_NONE

LIBUSB_LOG_LEVEL_ERROR

LIBUSB_LOG_LEVEL_WARNING

LIBUSB_LOG_LEVEL_INFO

LIBUSB_LOG_LEVEL_DEBUG

31.1.4 Function Documentation

void API_EXPORTED libusb_set_debug (libusb_context * *ctx*, int *level*)

Set log message verbosity.

The default level is LIBUSB_LOG_LEVEL_NONE, which means no messages are ever printed. If you choose to increase the message verbosity level, ensure that your application does not close the stdout/stderr file descriptors.

You are advised to use level LIBUSB_LOG_LEVEL_WARNING. libusb is conservative with its message logging and most of the time, will only log messages that explain error conditions and other oddities. This will help you debug your software.

If the LIBUSB_DEBUG environment variable was set when libusb was initialized, this function does nothing: the message verbosity is fixed to the value in the environment variable.

If libusb was compiled without any message logging, this function does nothing: you'll never get any messages.

If libusb was compiled with verbose debug message logging, this function does nothing: you'll always get messages from all levels.

Parameters

ctx	the context to operate on, or NULL for the default context
level	debug level to set

int API_EXPORTED libusb_init (libusb_context ** *context*)

Initialize libusb.

This function must be called before calling any other libusb function.

If you do not provide an output location for a context pointer, a default context will be created. If there was already a default context, it will be reused (and nothing will be initialized/reinitialized).

Parameters

context	Optional output location for context pointer. Only valid on return code 0.

Returns

0 on success, or a LIBUSB_ERROR code on failure

See Also

libusb_contexts

void API_EXPORTED libusb_exit (struct libusb_context * *ctx*)

Deinitialize libusb.

Should be called after closing all open devices and before your application terminates.

Parameters

ctx	the context to deinitialize, or NULL for the default context

31.2 Device handling and enumeration

The functionality documented below is designed to help with the following operations:

Typedefs

- typedef struct libusb_device libusb_device

  Structure representing a USB device detected on the system.
- typedef struct libusb_device_handle libusb_device_handle

  Structure representing a handle on a USB device.

Enumerations

- enum libusb_speed { LIBUSB_SPEED_UNKNOWN = 0, LIBUSB_SPEED_LOW = 1, LIBUSB_SPEED_FULL = 2, LIBUSB_SPEED_HIGH = 3, LIBUSB_SPEED_SUPER = 4 }

  Speed codes.
- enum libusb_supported_speed { LIBUSB_LOW_SPEED_OPERATION = 1, LIBUSB_FULL_SPEED_OPERATION = 2, LIBUSB_HIGH_SPEED_OPERATION = 4, LIBUSB_SUPER_SPEED_OPERATION = 8 }

  Supported speeds (wSpeedSupported) bitfield.
- enum libusb_usb_2_0_extension_attributes { LIBUSB_BM_LPM_SUPPORT = 2 }

  Masks for the bits of the bmAttributes field of the USB 2.0 Extension descriptor.
- enum libusb_ss_usb_device_capability_attributes { LIBUSB_BM_LTM_SUPPORT = 2 }

  Masks for the bits of the bmAttributes field field of the SuperSpeed USB Device Capability descriptor.
- enum libusb_bos_type { LIBUSB_BT_WIRELESS_USB_DEVICE_CAPABILITY = 1, LIBUSB_BT_USB_2_0_EXTENSION = 2, LIBUSB_BT_SS_USB_DEVICE_CAPABILITY = 3, LIBUSB_BT_CONTAINER_ID = 4 }

  USB capability types.

Functions

- ssize_t API_EXPORTED libusb_get_device_list (libusb_context *ctx, libusb_device ***list)

  Returns a list of USB devices currently attached to the system.
- void API_EXPORTED libusb_free_device_list (libusb_device **list, int unref_devices)

  Frees a list of devices previously discovered using libusb_get_device_list().
- uint8_t API_EXPORTED libusb_get_bus_number (libusb_device *dev)

  Get the number of the bus that a device is connected to.
- uint8_t API_EXPORTED libusb_get_port_number (libusb_device *dev)

  Get the number of the port that a device is connected to.
- int API_EXPORTED libusb_get_port_numbers (libusb_device *dev, uint8_t *port_numbers, int port_numbers_len)

  Get the list of all port numbers from root for the specified device.
- int API_EXPORTED libusb_get_port_path (libusb_context *ctx, libusb_device *dev, uint8_t *port_numbers, uint8_t port_numbers_len)

  Deprecated please use libusb_get_port_numbers instead.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_get_parent (libusb_device *dev)

  Get the the parent from the specified device.
- uint8_t API_EXPORTED libusb_get_device_address (libusb_device *dev)

  Get the address of the device on the bus it is connected to.
- int API_EXPORTED libusb_get_device_speed (libusb_device *dev)

  Get the negotiated connection speed for a device.
- int API_EXPORTED libusb_get_max_packet_size (libusb_device *dev, unsigned char endpoint)

  Convenience function to retrieve the wMaxPacketSize value for a particular endpoint in the active device configuration.
- int API_EXPORTED libusb_get_max_iso_packet_size (libusb_device *dev, unsigned char endpoint)

  Calculate the maximum packet size which a specific endpoint is capable is sending or receiving in the duration of 1 microframe.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_ref_device (libusb_device *dev)

  Increment the reference count of a device.
- void API_EXPORTED libusb_unref_device (libusb_device *dev)

  Decrement the reference count of a device.
- int API_EXPORTED libusb_open (libusb_device *dev, libusb_device_handle **dev_handle)

Open a device and obtain a device handle.

- DEFAULT_VISIBILITY libusb_device_handle *LIBUSB_CALL libusb_open_device_with_vid_pid (libusb_context *ctx, uint16_t vendor_id, uint16_t product_id)

  Convenience function for finding a device with a particular `idVendor/idProduct` combination.
- void API_EXPORTED libusb_close (libusb_device_handle *dev_handle)

  Close a device handle.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_get_device (libusb_device_handle *dev_handle)

  Get the underlying device for a device handle.
- int API_EXPORTED libusb_get_configuration (libusb_device_handle *dev_handle, int *config)

  Determine the bConfigurationValue of the currently active configuration.
- int API_EXPORTED libusb_set_configuration (libusb_device_handle *dev_handle, int configuration)

  Set the active configuration for a device.
- int API_EXPORTED libusb_claim_interface (libusb_device_handle *dev_handle, int interface_number)

  Claim an interface on a given device handle.
- int API_EXPORTED libusb_release_interface (libusb_device_handle *dev_handle, int interface_number)

  Release an interface previously claimed with libusb_claim_interface().
- int API_EXPORTED libusb_set_interface_alt_setting (libusb_device_handle *dev_handle, int interface_number, int alternate_setting)

  Activate an alternate setting for an interface.
- int API_EXPORTED libusb_clear_halt (libusb_device_handle *dev_handle, unsigned char endpoint)

  Clear the halt/stall condition for an endpoint.
- int API_EXPORTED libusb_reset_device (libusb_device_handle *dev_handle)

  Perform a USB port reset to reinitialize a device.
- int API_EXPORTED libusb_kernel_driver_active (libusb_device_handle *dev_handle, int interface_number)

  Determine if a kernel driver is active on an interface.
- int API_EXPORTED libusb_detach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Detach a kernel driver from an interface.
- int API_EXPORTED libusb_attach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Re-attach an interface's kernel driver, which was previously detached using libusb_detach_kernel_driver().
- int API_EXPORTED libusb_set_auto_detach_kernel_driver (libusb_device_handle *dev_handle, int enable)

  Enable/disable libusb's automatic kernel driver detachment.

31.2.1 Detailed Description

The functionality documented below is designed to help with the following operations:

- Enumerating the USB devices currently attached to the system
- Choosing a device to operate from your software
- Opening and closing the chosen device

31.2.2 In a nutshell...

The description below really makes things sound more complicated than they actually are. The following sequence of function calls will be suitable for almost all scenarios and does not require you to have such a deep understanding of the resource management issues:

```
// discover devices
libusb_device **list;
libusb_device *found = NULL;
ssize_t cnt = libusb_get_device_list(NULL, &list);
ssize_t i = 0;
int err = 0;
if (cnt < 0)
        error();

for (i = 0; i < cnt; i++) {
        libusb_device *device = list[i];
        if (is_interesting(device)) {
                found = device;
                break;
        }
}

if (found) {
        libusb_device_handle *handle;

        err = libusb_open(found, &handle);
        if (err)
                error();
        // etc
}

libusb_free_device_list(list, 1);
```

The two important points:

- You asked libusb_free_device_list() to unreference the devices (2nd parameter)
- You opened the device before freeing the list and unreferencing the devices

If you ended up with a handle, you can now proceed to perform I/O on the device.

31.2.3 Devices and device handles

libusb has a concept of a USB device, represented by the libusb_device opaque type. A device represents a USB device that is currently or was previously connected to the system. Using a reference to a device, you can determine certain information about the device (e.g. you can read the descriptor data).

The libusb_get_device_list() function can be used to obtain a list of devices currently connected to the system. This is known as device discovery.

Just because you have a reference to a device does not mean it is necessarily usable. The device may have been unplugged, you may not have permission to operate such device, or another program or driver may be using the device.

When you've found a device that you'd like to operate, you must ask libusb to open the device using the libusb_open() function. Assuming success, libusb then returns you a *device handle* (a libusb_device_handle pointer). All "real" I/O operations then operate on the handle rather than the original device pointer.

31.2.4 Device discovery and reference counting

Device discovery (i.e. calling libusb_get_device_list()) returns a freshly-allocated list of devices. The list itself must be freed when you are done with it. libusb also needs to know when it is OK to free the contents of the list - the devices themselves.

To handle these issues, libusb provides you with two separate items:

- A function to free the list itself
- A reference counting system for the devices inside

New devices presented by the libusb_get_device_list() function all have a reference count of 1. You can increase and decrease reference count using libusb_ref_device() and libusb_unref_device(). A device is destroyed when its reference count reaches 0.

With the above information in mind, the process of opening a device can be viewed as follows:

1. Discover devices using libusb_get_device_list().
2. Choose the device that you want to operate, and call libusb_open().
3. Unref all devices in the discovered device list.
4. Free the discovered device list.

The order is important - you must not unreference the device before attempting to open it, because unreferencing it may destroy the device.

For convenience, the libusb_free_device_list() function includes a parameter to optionally unreference all the devices in the list before freeing the list itself. This combines steps 3 and 4 above.

As an implementation detail, libusb_open() actually adds a reference to the device in question. This is because the device remains available through the handle via libusb_get_device(). The reference is deleted during libusb_close().

31.2.5 Typedef Documentation

typedef struct libusb_device libusb_device

Structure representing a USB device detected on the system.

This is an opaque type for which you are only ever provided with a pointer, usually originating from libusb_get_device_list().

Certain operations can be performed on a device, but in order to do any I/O you will have to first obtain a device handle using libusb_open().

Devices are reference counted with libusb_ref_device() and libusb_unref_device(), and are freed when the reference count reaches 0. New devices presented by libusb_get_device_list() have a reference count of 1, and libusb_free_device_list() can optionally decrease the reference count on all devices in the list. libusb_open() adds another reference which is later destroyed by libusb_close().

typedef struct libusb_device_handle libusb_device_handle

Structure representing a handle on a USB device.

This is an opaque type for which you are only ever provided with a pointer, usually originating from libusb_open().

A device handle is used to perform I/O and other operations. When finished with a device handle, you should call libusb_close().

31.2.6 Enumeration Type Documentation

enum libusb_speed

Speed codes.

Indicates the speed at which the device is operating.

Enumerator

LIBUSB_SPEED_UNKNOWN The OS doesn't report or know the device speed.

LIBUSB_SPEED_LOW The device is operating at low speed (1.5MBit/s).

LIBUSB_SPEED_FULL The device is operating at full speed (12MBit/s).

LIBUSB_SPEED_HIGH The device is operating at high speed (480MBit/s).

LIBUSB_SPEED_SUPER The device is operating at super speed (5000MBit/s).

enum libusb_supported_speed

Supported speeds (wSpeedSupported) bitfield.

Indicates what speeds the device supports.

Enumerator

LIBUSB_LOW_SPEED_OPERATION Low speed operation supported (1.5MBit/s).

LIBUSB_FULL_SPEED_OPERATION Full speed operation supported (12MBit/s).

LIBUSB_HIGH_SPEED_OPERATION High speed operation supported (480MBit/s).

LIBUSB_SUPER_SPEED_OPERATION Superspeed operation supported (5000MBit/s).

enum libusb_usb_2_0_extension_attributes

Masks for the bits of the bmAttributes field of the USB 2.0 Extension descriptor.

Enumerator

LIBUSB_BM_LPM_SUPPORT Supports Link Power Management (LPM)

enum libusb_ss_usb_device_capability_attributes

Masks for the bits of the bmAttributes field field of the SuperSpeed USB Device Capability descriptor.

Enumerator

LIBUSB_BM_LTM_SUPPORT Supports Latency Tolerance Messages (LTM)

enum libusb_bos_type

USB capability types.

Enumerator

LIBUSB_BT_WIRELESS_USB_DEVICE_CAPABILITY Wireless USB device capability.

LIBUSB_BT_USB_2_0_EXTENSION USB 2.0 extensions.

LIBUSB_BT_SS_USB_DEVICE_CAPABILITY SuperSpeed USB device capability.

LIBUSB_BT_CONTAINER_ID Container ID type.

31.2.7 Function Documentation

ssize_t API_EXPORTED libusb_get_device_list (libusb_context * *ctx*, libusb_device * *list*)**

Returns a list of USB devices currently attached to the system.

This is your entry point into finding a USB device to operate.

You are expected to unreference all the devices when you are done with them, and then free the list with libusb_free_device_list(). Note that libusb_free_device_list() can unref all the devices for you. Be careful not to unreference a device you are about to open until after you have opened it.

This return value of this function indicates the number of devices in the resultant list. The list is actually one element larger, as it is NULL-terminated.

Parameters

ctx	the context to operate on, or NULL for the default context
list	output location for a list of devices. Must be later freed with libusb_free_device_list().

Returns

the number of devices in the outputted list, or any libusb_error according to errors encountered by the backend.

void API_EXPORTED libusb_free_device_list (libusb_device ** *list*, int *unref_devices*)

Frees a list of devices previously discovered using libusb_get_device_list().

If the unref_devices parameter is set, the reference count of each device in the list is decremented by 1.

Parameters

list	the list to free
unref_devices	whether to unref the devices in the list

uint8_t API_EXPORTED libusb_get_bus_number (libusb_device * *dev*)

Get the number of the bus that a device is connected to.

Parameters

dev	a device

Returns

the bus number

uint8_t API_EXPORTED libusb_get_port_number (libusb_device * *dev*)

Get the number of the port that a device is connected to.

Unless the OS does something funky, or you are hot-plugging USB extension cards, the port number returned by this call is usually guaranteed to be uniquely tied to a physical port, meaning that different devices plugged on the same physical port should return the same port number.

But outside of this, there is no guarantee that the port number returned by this call will remain the same, or even match the order in which ports have been numbered by the HUB/HCD manufacturer.

Parameters

dev	a device

Returns

the port number (0 if not available)

int API_EXPORTED libusb_get_port_numbers (libusb_device * *dev*, uint8_t * *port_numbers*, int *port_numbers_len*)

Get the list of all port numbers from root for the specified device.

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102

Parameters

dev	a device
port_numbers	the array that should contain the port numbers
port_numbers_-len	the maximum length of the array. As per the USB 3.0 specs, the current maximum limit for the depth is 7.

Returns

the number of elements filled
LIBUSB_ERROR_OVERFLOW if the array is too small

int API_EXPORTED libusb_get_port_path (libusb_context * *ctx,* libusb_device * *dev,* uint8_t * *port_numbers,* uint8_t *port_numbers_len*)

Deprecated please use libusb_get_port_numbers instead.

DEFAULT_VISIBILITY libusb_device* LIBUSB_CALL libusb_get_parent (libusb_device * *dev*)

Get the the parent from the specified device.

Parameters

dev	a device

Returns

the device parent or NULL if not available You should issue a libusb_get_device_list() before calling this function and make sure that you only access the parent before issuing libusb_free_device_list(). The reason is that libusb currently does not maintain a permanent list of device instances, and therefore can only guarantee that parents are fully instantiated within a libusb_get_device_list() - libusb_free_device_list() block.

uint8_t API_EXPORTED libusb_get_device_address (libusb_device * *dev*)

Get the address of the device on the bus it is connected to.

Parameters

dev	a device

Returns

the device address

int API_EXPORTED libusb_get_device_speed (libusb_device * *dev*)

Get the negotiated connection speed for a device.

Parameters

dev	a device

Returns

a libusb_speed code, where LIBUSB_SPEED_UNKNOWN means that the OS doesn't know or doesn't support returning the negotiated speed.

int API_EXPORTED libusb_get_max_packet_size (libusb_device * *dev,* unsigned char *endpoint*)

Convenience function to retrieve the wMaxPacketSize value for a particular endpoint in the active device configuration.

This function was originally intended to be of assistance when setting up isochronous transfers, but a design mistake resulted in this function instead. It simply returns the wMaxPacketSize value without considering its contents. If you're dealing with isochronous transfers, you probably want libusb_get_max_iso_packet_size() instead.

Parameters

dev	a device
endpoint	address of the endpoint in question

Returns

the wMaxPacketSize value
LIBUSB_ERROR_NOT_FOUND if the endpoint does not exist
LIBUSB_ERROR_OTHER on other failure

int API_EXPORTED libusb_get_max_iso_packet_size (libusb_device * *dev*, unsigned char *endpoint*)

Calculate the maximum packet size which a specific endpoint is capable is sending or receiving in the duration of 1 microframe.

Only the active configuration is examined. The calculation is based on the wMaxPacketSize field in the endpoint descriptor as described in section 9.6.6 in the USB 2.0 specifications.

If acting on an isochronous or interrupt endpoint, this function will multiply the value found in bits 0:10 by the number of transactions per microframe (determined by bits 11:12). Otherwise, this function just returns the numeric value found in bits 0:10.

This function is useful for setting up isochronous transfers, for example you might pass the return value from this function to libusb_set_iso_packet_lengths() in order to set the length field of every isochronous packet in a transfer.

Since v1.0.3.

Parameters

dev	a device
endpoint	address of the endpoint in question

Returns

the maximum packet size which can be sent/received on this endpoint
LIBUSB_ERROR_NOT_FOUND if the endpoint does not exist
LIBUSB_ERROR_OTHER on other failure

DEFAULT_VISIBILITY libusb_device* LIBUSB_CALL libusb_ref_device (libusb_device * *dev*)

Increment the reference count of a device.

Parameters

dev	the device to reference

Returns

the same device

void API_EXPORTED libusb_unref_device (libusb_device * *dev*)

Decrement the reference count of a device.

If the decrement operation causes the reference count to reach zero, the device shall be destroyed.

Parameters

dev	the device to unreference

int API_EXPORTED libusb_open (libusb_device * *dev*, libusb_device_handle ** *dev_handle*)

Open a device and obtain a device handle.

A handle allows you to perform I/O on the device in question.

Internally, this function adds a reference to the device and makes it available to you through libusb_get_device(). This reference is removed during libusb_close().

This is a non-blocking function; no requests are sent over the bus.

Parameters

dev	the device to open
dev_handle	output location for the returned device handle pointer. Only populated when the return code is 0.

Returns

0 on success
LIBUSB_ERROR_NO_MEM on memory allocation failure
LIBUSB_ERROR_ACCESS if the user has insufficient permissions
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

DEFAULT_VISIBILITY libusb_device_handle* LIBUSB_CALL libusb_open_device_with_vid_pid (libusb_context * *ctx,* uint16_t *vendor_id,* uint16_t *product_id*)

Convenience function for finding a device with a particular `idVendor/idProduct` combination.

This function is intended for those scenarios where you are using libusb to knock up a quick test application - it allows you to avoid calling libusb_get_device_list() and worrying about traversing/freeing the list.

This function has limitations and is hence not intended for use in real applications: if multiple devices have the same IDs it will only give you the first one, etc.

Parameters

ctx	the context to operate on, or NULL for the default context
vendor_id	the idVendor value to search for
product_id	the idProduct value to search for

Returns

a device handle for the first found device, or NULL on error or if the device could not be found.

void API_EXPORTED libusb_close (libusb_device_handle * *dev_handle*)

Close a device handle.

Should be called on all open handles before your application exits.

Internally, this function destroys the reference that was added by libusb_open() on the given device.

This is a non-blocking function; no requests are sent over the bus.

Parameters

dev_handle	the device handle to close

DEFAULT_VISIBILITY libusb_device* LIBUSB_CALL libusb_get_device (libusb_device_handle * *dev_handle*)

Get the underlying device for a device handle.

This function does not modify the reference count of the returned device, so do not feel compelled to unreference it when you are done.

Parameters

dev_handle	a device handle

Returns

the underlying device

int API_EXPORTED libusb_get_configuration (libusb_device_handle * *dev_handle,* int * *config*)

Determine the bConfigurationValue of the currently active configuration.

You could formulate your own control request to obtain this information, but this function has the advantage that it may be able to retrieve the information from operating system caches (no I/O involved).

If the OS does not cache this information, then this function will block while a control transfer is submitted to retrieve the information.

This function will return a value of 0 in the `config` output parameter if the device is in unconfigured state.

Parameters

dev_handle	a device handle
config	output location for the bConfigurationValue of the active configuration (only valid for return code 0)

Returns

0 on success
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

int API_EXPORTED libusb_set_configuration (libusb_device_handle * *dev_handle,* int *configuration*)

Set the active configuration for a device.

The operating system may or may not have already set an active configuration on the device. It is up to your application to ensure the correct configuration is selected before you attempt to claim interfaces and perform other operations.

If you call this function on a device already configured with the selected configuration, then this function will act as a lightweight device reset: it will issue a SET_CONFIGURATION request using the current configuration, causing most USB-related device state to be reset (altsetting reset to zero, endpoint halts cleared, toggles reset).

You cannot change/reset configuration if your application has claimed interfaces. It is advised to set the desired configuration before claiming interfaces.

Alternatively you can call libusb_release_interface() first. Note if you do things this way you must ensure that auto_-detach_kernel_driver for `dev` is 0, otherwise the kernel driver will be re-attached when you release the interface(s).

You cannot change/reset configuration if other applications or drivers have claimed interfaces.

A configuration value of -1 will put the device in unconfigured state. The USB specifications state that a configuration value of 0 does this, however buggy devices exist which actually have a configuration 0.

You should always use this function rather than formulating your own SET_CONFIGURATION control request. This is because the underlying operating system needs to know when such changes happen.

This is a blocking function.

Parameters

dev_handle	a device handle
configuration	the bConfigurationValue of the configuration you wish to activate, or -1 if you wish to put the device in an unconfigured state

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the requested configuration does not exist
LIBUSB_ERROR_BUSY if interfaces are currently claimed
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

See Also

libusb_set_auto_detach_kernel_driver()

int API_EXPORTED libusb_claim_interface (libusb_device_handle * *dev_handle,* int *interface_number*)

Claim an interface on a given device handle.

You must claim the interface you wish to use before you can perform I/O on any of its endpoints.

It is legal to attempt to claim an already-claimed interface, in which case libusb just returns 0 without doing anything.

If auto_detach_kernel_driver is set to 1 for `dev`, the kernel driver will be detached if necessary, on failure the detach error is returned.

Claiming of interfaces is a purely logical operation; it does not cause any requests to be sent over the bus. Interface claiming is used to instruct the underlying operating system that your application wishes to take ownership of the interface.

This is a non-blocking function.

Parameters

dev_handle	a device handle
interface_number	the `bInterfaceNumber` of the interface you wish to claim

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the requested interface does not exist
LIBUSB_ERROR_BUSY if another program or driver has claimed the interface
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
a LIBUSB_ERROR code on other failure

See Also

libusb_set_auto_detach_kernel_driver()

int API_EXPORTED libusb_release_interface (libusb_device_handle * *dev_handle*, int *interface_number*)

Release an interface previously claimed with libusb_claim_interface().

You should release all claimed interfaces before closing a device handle.

This is a blocking function. A SET_INTERFACE control request will be sent to the device, resetting interface state to the first alternate setting.

If auto_detach_kernel_driver is set to 1 for `dev`, the kernel driver will be re-attached after releasing the interface.

Parameters

dev_handle	a device handle
interface_number	the `bInterfaceNumber` of the previously-claimed interface

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the interface was not claimed
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

See Also

libusb_set_auto_detach_kernel_driver()

int API_EXPORTED libusb_set_interface_alt_setting (libusb_device_handle * *dev_handle*, int *interface_number*, int *alternate_setting*)

Activate an alternate setting for an interface.

The interface must have been previously claimed with libusb_claim_interface().

You should always use this function rather than formulating your own SET_INTERFACE control request. This is because the underlying operating system needs to know when such changes happen.

This is a blocking function.

Parameters

dev_handle	a device handle
interface_number	the `bInterfaceNumber` of the previously-claimed interface
alternate_setting	the `bAlternateSetting` of the alternate setting to activate

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the interface was not claimed, or the requested alternate setting does not exist
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

int **API_EXPORTED** libusb_clear_halt (**libusb_device_handle** * *dev_handle,* unsigned char *endpoint*)

Clear the halt/stall condition for an endpoint.

Endpoints with halt status are unable to receive or transmit data until the halt condition is stalled.

You should cancel all pending transfers before attempting to clear the halt condition.

This is a blocking function.

Parameters

dev_handle	a device handle
endpoint	the endpoint to clear halt status

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the endpoint does not exist
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
another LIBUSB_ERROR code on other failure

int API_EXPORTED libusb_reset_device (libusb_device_handle * *dev_handle*)

Perform a USB port reset to reinitialize a device.

The system will attempt to restore the previous configuration and alternate settings after the reset has completed.

If the reset fails, the descriptors change, or the previous state cannot be restored, the device will appear to be disconnected and reconnected. This means that the device handle is no longer valid (you should close it) and rediscover the device. A return code of LIBUSB_ERROR_NOT_FOUND indicates when this is the case.

This is a blocking function which usually incurs a noticeable delay.

Parameters

dev_handle	a handle of the device to reset

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if re-enumeration is required, or if the device has been disconnected
another LIBUSB_ERROR code on other failure

int API_EXPORTED libusb_kernel_driver_active (libusb_device_handle * *dev_handle*, int *interface_number*)

Determine if a kernel driver is active on an interface.

If a kernel driver is active, you cannot claim the interface, and libusb will be unable to perform I/O.

This functionality is not available on Windows.

Parameters

dev_handle	a device handle
interface_number	the interface to check

Returns

0 if no kernel driver is active
1 if a kernel driver is active
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_NOT_SUPPORTED on platforms where the functionality is not available
another LIBUSB_ERROR code on other failure

See Also

libusb_detach_kernel_driver()

int API_EXPORTED libusb_detach_kernel_driver (libusb_device_handle * *dev_handle*, int *interface_number*)

Detach a kernel driver from an interface.

If successful, you will then be able to claim the interface and perform I/O.

This functionality is not available on Darwin or Windows.

Note that libusb itself also talks to the device through a special kernel driver, if this driver is already attached to the device, this call will not detach it and return LIBUSB_ERROR_NOT_FOUND.

Parameters

dev_handle	a device handle
interface_number	the interface to detach the driver from

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if no kernel driver was active
LIBUSB_ERROR_INVALID_PARAM if the interface does not exist
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_NOT_SUPPORTED on platforms where the functionality is not available
another LIBUSB_ERROR code on other failure

See Also

libusb_kernel_driver_active()

int API_EXPORTED libusb_attach_kernel_driver (libusb_device_handle * *dev_handle*, int *interface_number*)

Re-attach an interface's kernel driver, which was previously detached using libusb_detach_kernel_driver().

This call is only effective on Linux and returns LIBUSB_ERROR_NOT_SUPPORTED on all other platforms.

This functionality is not available on Darwin or Windows.

Parameters

dev_handle	a device handle
interface_number	the interface to attach the driver from

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if no kernel driver was active
LIBUSB_ERROR_INVALID_PARAM if the interface does not exist
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_NOT_SUPPORTED on platforms where the functionality is not available
LIBUSB_ERROR_BUSY if the driver cannot be attached because the interface is claimed by a program or driver
another LIBUSB_ERROR code on other failure

See Also

libusb_kernel_driver_active()

int API_EXPORTED libusb_set_auto_detach_kernel_driver (libusb_device_handle * *dev_handle*, int *enable*)

Enable/disable libusb's automatic kernel driver detachment.

When this is enabled libusb will automatically detach the kernel driver on an interface when claiming the interface, and attach it when releasing the interface.

Automatic kernel driver detachment is disabled on newly opened device handles by default.

On platforms which do not have LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER this function will return LIBUSB_ERROR_NOT_SUPPORTED, and libusb will continue as if this function was never called.

Parameters

dev_handle	a device handle
enable	whether to enable or disable auto kernel driver detachment

Returns

LIBUSB_SUCCESS on success
LIBUSB_ERROR_NOT_SUPPORTED on platforms where the functionality is not available

See Also

libusb_claim_interface()
libusb_release_interface()
libusb_set_configuration()

31.3 Miscellaneous

Macros

- #define LIBUSB_CALL

  libusb's Windows calling convention.

- #define LIBUSB_API_VERSION 0x01000105

  libusb's API version.

- #define libusb_le16_to_cpu libusb_cpu_to_le16

  Convert a 16-bit value from little-endian to host-endian format.

Enumerations

- enum libusb_standard_request {
  LIBUSB_REQUEST_GET_STATUS = 0x00, LIBUSB_REQUEST_CLEAR_FEATURE = 0x01, LIBUSB_REQUEST_SET_FEATURE = 0x03, LIBUSB_REQUEST_SET_ADDRESS = 0x05,
  LIBUSB_REQUEST_GET_DESCRIPTOR = 0x06, LIBUSB_REQUEST_SET_DESCRIPTOR = 0x07, LIBUSB_REQUEST_GET_CONFIGURATION = 0x08, LIBUSB_REQUEST_SET_CONFIGURATION = 0x09,
  LIBUSB_REQUEST_GET_INTERFACE = 0x0A, LIBUSB_REQUEST_SET_INTERFACE = 0x0B, LIBUSB_REQUEST_SYNCH_FRAME = 0x0C, LIBUSB_REQUEST_SET_SEL = 0x30,
  LIBUSB_SET_ISOCH_DELAY = 0x31 }

  Standard requests, as defined in table 9-5 of the USB 3.0 specifications.

- enum libusb_request_type { LIBUSB_REQUEST_TYPE_STANDARD = (0x00 << 5), LIBUSB_REQUEST_TYPE_CLASS = (0x01 << 5), LIBUSB_REQUEST_TYPE_VENDOR = (0x02 << 5), LIBUSB_REQUEST_TYPE_RESERVED = (0x03 << 5) }

  Request type bits of the bmRequestType field in control transfers.

- enum libusb_request_recipient { LIBUSB_RECIPIENT_DEVICE = 0x00, LIBUSB_RECIPIENT_INTERFACE = 0x01, LIBUSB_RECIPIENT_ENDPOINT = 0x02, LIBUSB_RECIPIENT_OTHER = 0x03 }

  Recipient bits of the bmRequestType field in control transfers.

- enum libusb_error {
  LIBUSB_SUCCESS = 0, LIBUSB_ERROR_IO = -1, LIBUSB_ERROR_INVALID_PARAM = -2, LIBUSB_ERROR_ACCESS = -3,
  LIBUSB_ERROR_NO_DEVICE = -4, LIBUSB_ERROR_NOT_FOUND = -5, LIBUSB_ERROR_BUSY = -6, LIBUSB_ERROR_TIMEOUT = -7,
  LIBUSB_ERROR_OVERFLOW = -8, LIBUSB_ERROR_PIPE = -9, LIBUSB_ERROR_INTERRUPTED = -10, LIBUSB_ERROR_NO_MEM = -11,
  LIBUSB_ERROR_NOT_SUPPORTED = -12, LIBUSB_ERROR_OTHER = -99 }

  Error codes.

- enum libusb_capability { LIBUSB_CAP_HAS_CAPABILITY = 0x0000, LIBUSB_CAP_HAS_HOTPLUG = 0x0001, LIBUSB_CAP_HAS_HID_ACCESS = 0x0100, LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER = 0x0101 }

  Capabilities supported by an instance of libusb on the current running platform.

Functions

- int API_EXPORTED libusb_has_capability (uint32_t capability)

  Check at runtime if the loaded library has a given capability.

- DEFAULT_VISIBILITY const char
  *LIBUSB_CALL libusb_error_name (int error_code)

  Returns a constant NULL-terminated string with the ASCII name of a libusb error or transfer status code.

- DEFAULT_VISIBILITY const
  struct libusb_version
  *LIBUSB_CALL libusb_get_version (void)

  Returns a pointer to const struct libusb_version with the version (major, minor, micro, nano and rc) of the running library.

- int API_EXPORTED libusb_setlocale (const char *locale)

  Set the language, and only the language, not the encoding! used for translatable libusb messages.

- DEFAULT_VISIBILITY const char
  *LIBUSB_CALL libusb_strerror (enum libusb_error errcode)

  Returns a constant string with a short description of the given error code, this description is intended for displaying to the end user and will be in the language set by libusb_setlocale().

31.3.1 Detailed Description

31.3.2 Macro Definition Documentation

#define LIBUSB_CALL

libusb's Windows calling convention.

Under Windows, the selection of available compilers and configurations means that, unlike other platforms, there is not *one true calling convention* (calling convention: the manner in which parameters are passed to functions in the generated assembly code).

Matching the Windows API itself, libusb uses the WINAPI convention (which translates to the `stdcall` convention) and guarantees that the library is compiled in this way. The public header file also includes appropriate annotations so that your own software will use the right convention, even if another convention is being used by default within your codebase.

The one consideration that you must apply in your software is to mark all functions which you use as libusb callbacks with this LIBUSB_CALL annotation, so that they too get compiled for the correct calling convention.

On non-Windows operating systems, this macro is defined as nothing. This means that you can apply it to your code without worrying about cross-platform compatibility.

#define LIBUSB_API_VERSION 0x01000105

libusb's API version.

Since version 1.0.13, to help with feature detection, libusb defines a LIBUSB_API_VERSION macro that gets increased every time there is a significant change to the API, such as the introduction of a new call, the definition of a new macro/enum member, or any other element that libusb applications may want to detect at compilation time.

The macro is typically used in an application as follows:

```
#if defined(LIBUSB_API_VERSION) && (LIBUSB_API_VERSION >= 0x01001234)
// Use one of the newer features from the libusb API
#endif
```

Internally, LIBUSB_API_VERSION is defined as follows: (libusb major << 24) | (libusb minor << 16) | (16 bit incremental)

#define libusb_le16_to_cpu libusb_cpu_to_le16

Convert a 16-bit value from little-endian to host-endian format.

On little endian systems, this function does nothing. On big endian systems, the bytes are swapped.

Parameters

x	the little-endian value to convert

Returns

the value in host-endian byte order

31.3.3 Enumeration Type Documentation

enum libusb_standard_request

Standard requests, as defined in table 9-5 of the USB 3.0 specifications.

Enumerator

LIBUSB_REQUEST_GET_STATUS Request status of the specific recipient.

LIBUSB_REQUEST_CLEAR_FEATURE Clear or disable a specific feature.

LIBUSB_REQUEST_SET_FEATURE Set or enable a specific feature.

LIBUSB_REQUEST_SET_ADDRESS Set device address for all future accesses.

LIBUSB_REQUEST_GET_DESCRIPTOR Get the specified descriptor.

LIBUSB_REQUEST_SET_DESCRIPTOR Used to update existing descriptors or add new descriptors.

LIBUSB_REQUEST_GET_CONFIGURATION Get the current device configuration value.

LIBUSB_REQUEST_SET_CONFIGURATION Set device configuration.

LIBUSB_REQUEST_GET_INTERFACE Return the selected alternate setting for the specified interface.

LIBUSB_REQUEST_SET_INTERFACE Select an alternate interface for the specified interface.

LIBUSB_REQUEST_SYNCH_FRAME Set then report an endpoint's synchronization frame.

LIBUSB_REQUEST_SET_SEL Sets both the U1 and U2 Exit Latency.

LIBUSB_SET_ISOCH_DELAY Delay from the time a host transmits a packet to the time it is received by the device.

enum libusb_request_type

Request type bits of the bmRequestType field in control transfers.

Enumerator

LIBUSB_REQUEST_TYPE_STANDARD Standard.

LIBUSB_REQUEST_TYPE_CLASS Class.

LIBUSB_REQUEST_TYPE_VENDOR Vendor.

LIBUSB_REQUEST_TYPE_RESERVED Reserved.

enum libusb_request_recipient

Recipient bits of the bmRequestType field in control transfers.

Values 4 through 31 are reserved.

Enumerator

LIBUSB_RECIPIENT_DEVICE Device.

LIBUSB_RECIPIENT_INTERFACE Interface.

LIBUSB_RECIPIENT_ENDPOINT Endpoint.

LIBUSB_RECIPIENT_OTHER Other.

enum libusb_error

Error codes.

Most libusb functions return 0 on success or one of these codes on failure. You can call libusb_error_name() to retrieve a string representation of an error code or libusb_strerror() to get an end-user suitable description of an error code.

Enumerator

LIBUSB_SUCCESS Success (no error)

LIBUSB_ERROR_IO Input/output error.

LIBUSB_ERROR_INVALID_PARAM Invalid parameter.

LIBUSB_ERROR_ACCESS Access denied (insufficient permissions)

LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)

LIBUSB_ERROR_NOT_FOUND Entity not found.

LIBUSB_ERROR_BUSY Resource busy.

LIBUSB_ERROR_TIMEOUT Operation timed out.

LIBUSB_ERROR_OVERFLOW Overflow.

LIBUSB_ERROR_PIPE Pipe error.

LIBUSB_ERROR_INTERRUPTED System call interrupted (perhaps due to signal)

LIBUSB_ERROR_NO_MEM Insufficient memory.

LIBUSB_ERROR_NOT_SUPPORTED Operation not supported or unimplemented on this platform.

LIBUSB_ERROR_OTHER Other error.

enum libusb_capability

Capabilities supported by an instance of libusb on the current running platform.

Test if the loaded library supports a given capability by calling libusb_has_capability().

Enumerator

LIBUSB_CAP_HAS_CAPABILITY The libusb_has_capability() API is available.

LIBUSB_CAP_HAS_HOTPLUG Hotplug support is available on this platform.

LIBUSB_CAP_HAS_HID_ACCESS The library can access HID devices without requiring user intervention. Note that before being able to actually access an HID device, you may still have to call additional libusb functions such as libusb_detach_kernel_driver().

LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER The library supports detaching of the default USB driver, using libusb_detach_kernel_driver(), if one is set by the OS kernel.

31.3.4 Function Documentation

int API_EXPORTED libusb_has_capability (uint32_t *capability*)

Check at runtime if the loaded library has a given capability.

This call should be performed after libusb_init(), to ensure the backend has updated its capability set.

Parameters

capability	the libusb_capability to check for

Returns

nonzero if the running library has the capability, 0 otherwise

DEFAULT_VISIBILITY const char* LIBUSB_CALL libusb_error_name (int *error_code*)

Returns a constant NULL-terminated string with the ASCII name of a libusb error or transfer status code.

The caller must not free() the returned string.

Parameters

error_code	The libusb_error or libusb_transfer_status code to return the name of.

Returns

The error name, or the string **UNKNOWN** if the value of error_code is not a known error / status code.

DEFAULT_VISIBILITY const struct libusb_version* LIBUSB_CALL libusb_get_version (void)

Returns a pointer to const struct libusb_version with the version (major, minor, micro, nano and rc) of the running library.

int API_EXPORTED libusb_setlocale (const char * *locale*)

Set the language, and only the language, not the encoding! used for translatable libusb messages.

This takes a locale string in the default setlocale format: lang[-region] or lang[_country_region][.codeset]. Only the lang part of the string is used, and only 2 letter ISO 639-1 codes are accepted for it, such as "de". The optional region, country_region or codeset parts are ignored. This means that functions which return translatable strings will NOT honor the specified encoding. All strings returned are encoded as UTF-8 strings.

If libusb_setlocale() is not called, all messages will be in English.

The following functions return translatable strings: libusb_strerror(). Note that the libusb log messages controlled through libusb_set_debug() are not translated, they are always in English.

For POSIX UTF-8 environments if you want libusb to follow the standard locale settings, call libusb_setlocale(setlocale(-LC_MESSAGES, NULL)), after your app has done its locale setup.

Parameters

locale	locale-string in the form of lang[_country_region][.codeset] or lang[-region], where lang is a 2 letter ISO 639-1 code

Returns

LIBUSB_SUCCESS on success
LIBUSB_ERROR_INVALID_PARAM if the locale doesn't meet the requirements
LIBUSB_ERROR_NOT_FOUND if the requested language is not supported
a LIBUSB_ERROR code on other errors

DEFAULT_VISIBILITY const char* LIBUSB_CALL libusb_strerror (enum libusb_error *errcode*)

Returns a constant string with a short description of the given error code, this description is intended for displaying to the end user and will be in the language set by libusb_setlocale().

The returned string is encoded in UTF-8.

The messages always start with a capital letter and end without any dot. The caller must not free() the returned string.

Parameters

errcode	the error code whose description is desired

Returns

a short description of the error code in UTF-8 encoding

31.4 USB descriptors

This page details how to examine the various standard USB descriptors for detected devices.

Data Structures

- struct libusb_device_descriptor

  A structure representing the standard USB device descriptor.
- struct libusb_endpoint_descriptor

  A structure representing the standard USB endpoint descriptor.
- struct libusb_interface_descriptor

  A structure representing the standard USB interface descriptor.
- struct libusb_interface

  A collection of alternate settings for a particular USB interface.
- struct libusb_config_descriptor

  A structure representing the standard USB configuration descriptor.
- struct libusb_ss_endpoint_companion_descriptor

  A structure representing the superspeed endpoint companion descriptor.
- struct libusb_bos_dev_capability_descriptor

  A generic representation of a BOS Device Capability descriptor.
- struct libusb_bos_descriptor

  A structure representing the Binary Device Object Store (BOS) descriptor.
- struct libusb_usb_2_0_extension_descriptor

  A structure representing the USB 2.0 Extension descriptor This descriptor is documented in section 9.6.2.1 of the USB 3.0 specification.
- struct libusb_ss_usb_device_capability_descriptor

  A structure representing the SuperSpeed USB Device Capability descriptor This descriptor is documented in section 9.6.2.2 of the USB 3.0 specification.
- struct libusb_container_id_descriptor

  A structure representing the Container ID descriptor.

Enumerations

- enum libusb_class_code {
  LIBUSB_CLASS_PER_INTERFACE = 0, LIBUSB_CLASS_AUDIO = 1, LIBUSB_CLASS_COMM = 2, LIBUSB_CLASS_HID = 3,
  LIBUSB_CLASS_PHYSICAL = 5, LIBUSB_CLASS_PRINTER = 7, LIBUSB_CLASS_PTP = 6, LIBUSB_CLASS_IMAGE = 6,
  LIBUSB_CLASS_MASS_STORAGE = 8, LIBUSB_CLASS_HUB = 9, LIBUSB_CLASS_DATA = 10, LIBUSB_CLASS_SMART_CARD = 0x0b,
  LIBUSB_CLASS_CONTENT_SECURITY = 0x0d, LIBUSB_CLASS_VIDEO = 0x0e, LIBUSB_CLASS_PERSONAL_HEALTHCARE = 0x0f, LIBUSB_CLASS_DIAGNOSTIC_DEVICE = 0xdc,
  LIBUSB_CLASS_WIRELESS = 0xe0, LIBUSB_CLASS_APPLICATION = 0xfe, LIBUSB_CLASS_VENDOR_SPEC = 0xff }

  Device and/or Interface Class codes.
- enum libusb_descriptor_type {
  LIBUSB_DT_DEVICE = 0x01, LIBUSB_DT_CONFIG = 0x02, LIBUSB_DT_STRING = 0x03, LIBUSB_DT_INTERFACE = 0x04,
  LIBUSB_DT_ENDPOINT = 0x05, LIBUSB_DT_BOS = 0x0f, LIBUSB_DT_DEVICE_CAPABILITY = 0x10, LIBUSB_DT_HID = 0x21,
  LIBUSB_DT_REPORT = 0x22, LIBUSB_DT_PHYSICAL = 0x23, LIBUSB_DT_HUB = 0x29, LIBUSB_DT_SUPERSPEED_HUB = 0x2a,
  LIBUSB_DT_SS_ENDPOINT_COMPANION = 0x30 }

  Descriptor types as defined by the USB specification.
- enum libusb_endpoint_direction { LIBUSB_ENDPOINT_IN = 0x80, LIBUSB_ENDPOINT_OUT = 0x00 }

  Endpoint direction.
- enum libusb_transfer_type {
  LIBUSB_TRANSFER_TYPE_CONTROL = 0, LIBUSB_TRANSFER_TYPE_ISOCHRONOUS = 1, LIBUSB_TRANSFER_TYPE_BULK = 2, LIBUSB_TRANSFER_TYPE_INTERRUPT = 3,
  LIBUSB_TRANSFER_TYPE_BULK_STREAM = 4 }

  Endpoint transfer type.
- enum libusb_iso_sync_type { LIBUSB_ISO_SYNC_TYPE_NONE = 0, LIBUSB_ISO_SYNC_TYPE_ASYNC = 1, LIBUSB_ISO_SYNC_TYPE_ADAPTIVE = 2, LIBUSB_ISO_SYNC_TYPE_SYNC = 3 }

  Synchronization type for isochronous endpoints.
- enum libusb_iso_usage_type { LIBUSB_ISO_USAGE_TYPE_DATA = 0, LIBUSB_ISO_USAGE_TYPE_FEEDBACK = 1, LIBUSB_ISO_USAGE_TYPE_IMPLICIT = 2 }

  Usage type for isochronous endpoints.

Functions

- int API_EXPORTED libusb_get_device_descriptor (libusb_device *dev, struct libusb_device_descriptor *desc)

  Get the USB device descriptor for a given device.
- int API_EXPORTED libusb_get_active_config_descriptor (libusb_device *dev, struct libusb_config_descriptor **config)

  Get the USB configuration descriptor for the currently active configuration.
- int API_EXPORTED libusb_get_config_descriptor (libusb_device *dev, uint8_t config_index, struct libusb_config_descriptor **config)

  Get a USB configuration descriptor based on its index.
- int API_EXPORTED libusb_get_config_descriptor_by_value (libusb_device *dev, uint8_t bConfigurationValue, struct libusb_config_descriptor **config)

  Get a USB configuration descriptor with a specific bConfigurationValue.
- void API_EXPORTED libusb_free_config_descriptor (struct libusb_config_descriptor *config)

  Free a configuration descriptor obtained from libusb_get_active_config_descriptor() or libusb_get_config_descriptor().
- int API_EXPORTED libusb_get_ss_endpoint_companion_descriptor (struct libusb_context *ctx, const struct libusb_endpoint_descriptor *endpoint, struct libusb_ss_endpoint_companion_descriptor **ep_comp)

  Get an endpoints superspeed endpoint companion descriptor (if any)
- void API_EXPORTED libusb_free_ss_endpoint_companion_descriptor (struct libusb_ss_endpoint_companion_descriptor *ep_comp)

  Free a superspeed endpoint companion descriptor obtained from libusb_get_ss_endpoint_companion_descriptor().
- int API_EXPORTED libusb_get_bos_descriptor (libusb_device_handle *dev_handle, struct libusb_bos_descriptor **bos)

  Get a Binary Object Store (BOS) descriptor This is a BLOCKING function, which will send requests to the device.
- void API_EXPORTED libusb_free_bos_descriptor (struct libusb_bos_descriptor *bos)

  Free a BOS descriptor obtained from libusb_get_bos_descriptor().
- int API_EXPORTED libusb_get_usb_2_0_extension_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_usb_2_0_extension_descriptor **usb_2_0_extension)

  Get an USB 2.0 Extension descriptor.
- void API_EXPORTED libusb_free_usb_2_0_extension_descriptor (struct libusb_usb_2_0_extension_descriptor *usb_2_0_extension)

  Free a USB 2.0 Extension descriptor obtained from libusb_get_usb_2_0_extension_descriptor().
- int API_EXPORTED libusb_get_ss_usb_device_capability_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_ss_usb_device_capability_descriptor **ss_usb_device_cap)

  Get a SuperSpeed USB Device Capability descriptor.
- void API_EXPORTED libusb_free_ss_usb_device_capability_descriptor (struct libusb_ss_usb_device_capability_descriptor *ss_usb_device_cap)

  Free a SuperSpeed USB Device Capability descriptor obtained from libusb_get_ss_usb_device_capability_descriptor().
- int API_EXPORTED libusb_get_container_id_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_container_id_descriptor **container_id)

  Get a Container ID descriptor.
- void API_EXPORTED libusb_free_container_id_descriptor (struct libusb_container_id_descriptor *container_id)

  Free a Container ID descriptor obtained from libusb_get_container_id_descriptor().
- int API_EXPORTED libusb_get_string_descriptor_ascii (libusb_device_handle *dev_handle, uint8_t desc_index, unsigned char *data, int length)

  Retrieve a string descriptor in C style ASCII.

31.4.1 Detailed Description

This page details how to examine the various standard USB descriptors for detected devices.

31.4.2 Enumeration Type Documentation

enum libusb_class_code

Device and/or Interface Class codes.

Enumerator

LIBUSB_CLASS_PER_INTERFACE In the context of a device descriptor, this bDeviceClass value indicates that each interface specifies its own class information and all interfaces operate independently.

LIBUSB_CLASS_AUDIO Audio class.

LIBUSB_CLASS_COMM Communications class.

LIBUSB_CLASS_HID Human Interface Device class.

LIBUSB_CLASS_PHYSICAL Physical.

LIBUSB_CLASS_PRINTER Printer class.

LIBUSB_CLASS_PTP Image class.

LIBUSB_CLASS_IMAGE

LIBUSB_CLASS_MASS_STORAGE Mass storage class.

LIBUSB_CLASS_HUB Hub class.

LIBUSB_CLASS_DATA Data class.

LIBUSB_CLASS_SMART_CARD Smart Card.

LIBUSB_CLASS_CONTENT_SECURITY Content Security.

LIBUSB_CLASS_VIDEO Video.

LIBUSB_CLASS_PERSONAL_HEALTHCARE Personal Healthcare.

LIBUSB_CLASS_DIAGNOSTIC_DEVICE Diagnostic Device.

LIBUSB_CLASS_WIRELESS Wireless class.

LIBUSB_CLASS_APPLICATION Application class.

LIBUSB_CLASS_VENDOR_SPEC Class is vendor-specific.

enum libusb_descriptor_type

Descriptor types as defined by the USB specification.

Enumerator

LIBUSB_DT_DEVICE Device descriptor. See libusb_device_descriptor.

LIBUSB_DT_CONFIG Configuration descriptor. See libusb_config_descriptor.

LIBUSB_DT_STRING String descriptor.

LIBUSB_DT_INTERFACE Interface descriptor. See libusb_interface_descriptor.

LIBUSB_DT_ENDPOINT Endpoint descriptor. See libusb_endpoint_descriptor.

LIBUSB_DT_BOS BOS descriptor.

LIBUSB_DT_DEVICE_CAPABILITY Device Capability descriptor.

LIBUSB_DT_HID HID descriptor.

LIBUSB_DT_REPORT HID report descriptor.

LIBUSB_DT_PHYSICAL Physical descriptor.

LIBUSB_DT_HUB Hub descriptor.

LIBUSB_DT_SUPERSPEED_HUB SuperSpeed Hub descriptor.

LIBUSB_DT_SS_ENDPOINT_COMPANION SuperSpeed Endpoint Companion descriptor.

enum libusb_endpoint_direction

Endpoint direction.

Values for bit 7 of the endpoint address scheme.

Enumerator

LIBUSB_ENDPOINT_IN In: device-to-host.

LIBUSB_ENDPOINT_OUT Out: host-to-device.

enum libusb_transfer_type

Endpoint transfer type.

Values for bits 0:1 of the endpoint attributes field.

Enumerator

LIBUSB_TRANSFER_TYPE_CONTROL Control endpoint.

LIBUSB_TRANSFER_TYPE_ISOCHRONOUS Isochronous endpoint.

LIBUSB_TRANSFER_TYPE_BULK Bulk endpoint.

LIBUSB_TRANSFER_TYPE_INTERRUPT Interrupt endpoint.

LIBUSB_TRANSFER_TYPE_BULK_STREAM Stream endpoint.

enum libusb_iso_sync_type

Synchronization type for isochronous endpoints.

Values for bits 2:3 of the bmAttributes field in libusb_endpoint_descriptor.

Enumerator

LIBUSB_ISO_SYNC_TYPE_NONE No synchronization.
LIBUSB_ISO_SYNC_TYPE_ASYNC Asynchronous.
LIBUSB_ISO_SYNC_TYPE_ADAPTIVE Adaptive.
LIBUSB_ISO_SYNC_TYPE_SYNC Synchronous.

enum libusb_iso_usage_type

Usage type for isochronous endpoints.

Values for bits 4:5 of the bmAttributes field in libusb_endpoint_descriptor.

Enumerator

LIBUSB_ISO_USAGE_TYPE_DATA Data endpoint.
LIBUSB_ISO_USAGE_TYPE_FEEDBACK Feedback endpoint.
LIBUSB_ISO_USAGE_TYPE_IMPLICIT Implicit feedback Data endpoint.

31.4.3 Function Documentation

int API_EXPORTED libusb_get_device_descriptor (libusb_device * *dev*, struct libusb_device_descriptor * *desc*)

Get the USB device descriptor for a given device.

This is a non-blocking function; the device descriptor is cached in memory.

Note since libusb-1.0.16, LIBUSB_API_VERSION >= 0x01000102, this function always succeeds.

Parameters

dev	the device
desc	output location for the descriptor data

Returns

0 on success or a LIBUSB_ERROR code on failure

int API_EXPORTED libusb_get_active_config_descriptor (libusb_device * *dev*, struct libusb_config_descriptor ** *config*)

Get the USB configuration descriptor for the currently active configuration.

This is a non-blocking function which does not involve any requests being sent to the device.

Parameters

dev	a device
config	output location for the USB configuration descriptor. Only valid if 0 was returned. Must be freed with libusb_free_config_descriptor() after use.

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the device is in unconfigured state
another LIBUSB_ERROR code on error

See Also

libusb_get_config_descriptor

int API_EXPORTED libusb_get_config_descriptor (libusb_device * *dev*, uint8_t *config_index*, struct libusb_config_descriptor ** *config*)

Get a USB configuration descriptor based on its index.

This is a non-blocking function which does not involve any requests being sent to the device.

Parameters

dev	a device
config_index	the index of the configuration you wish to retrieve
config	output location for the USB configuration descriptor. Only valid if 0 was returned. Must be freed with libusb_free_config_descriptor() after use.

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the configuration does not exist
another LIBUSB_ERROR code on error

See Also

libusb_get_active_config_descriptor()
libusb_get_config_descriptor_by_value()

int API_EXPORTED libusb_get_config_descriptor_by_value (libusb_device * *dev*, uint8_t *bConfigurationValue*, struct libusb_config_descriptor ** *config*)

Get a USB configuration descriptor with a specific bConfigurationValue.

This is a non-blocking function which does not involve any requests being sent to the device.

Parameters

dev	a device
bConfiguration-Value	the bConfigurationValue of the configuration you wish to retrieve
config	output location for the USB configuration descriptor. Only valid if 0 was returned. Must be freed with libusb_free_config_descriptor() after use.

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the configuration does not exist
another LIBUSB_ERROR code on error

See Also

libusb_get_active_config_descriptor()
libusb_get_config_descriptor()

void API_EXPORTED libusb_free_config_descriptor (struct libusb_config_descriptor * *config*)

Free a configuration descriptor obtained from libusb_get_active_config_descriptor() or libusb_get_config_descriptor().

It is safe to call this function with a NULL config parameter, in which case the function simply returns.

Parameters

config	the configuration descriptor to free

int API_EXPORTED libusb_get_ss_endpoint_companion_descriptor (struct libusb_context * *ctx*, const struct libusb_endpoint_descriptor * *endpoint*, struct libusb_ss_endpoint_companion_descriptor ** *ep_comp*)

Get an endpoints superspeed endpoint companion descriptor (if any)

Parameters

ctx	the context to operate on, or NULL for the default context
endpoint	endpoint descriptor from which to get the superspeed endpoint companion descriptor
ep_comp	output location for the superspeed endpoint companion descriptor. Only valid if 0 was returned. Must be freed with libusb_free_ss_endpoint_companion_descriptor() after use.

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the configuration does not exist
another LIBUSB_ERROR code on error

void API_EXPORTED libusb_free_ss_endpoint_companion_descriptor (struct libusb_ss_endpoint_companion_descriptor * *ep_comp*)

Free a superspeed endpoint companion descriptor obtained from libusb_get_ss_endpoint_companion_descriptor().

It is safe to call this function with a NULL ep_comp parameter, in which case the function simply returns.

Parameters

ep_comp	the superspeed endpoint companion descriptor to free

int API_EXPORTED libusb_get_bos_descriptor (libusb_device_handle * *dev_handle,* struct libusb_bos_descriptor ** *bos*)

Get a Binary Object Store (BOS) descriptor This is a BLOCKING function, which will send requests to the device.

Parameters

dev_handle	the handle of an open libusb device
bos	output location for the BOS descriptor. Only valid if 0 was returned. Must be freed with libusb_free_bos_descriptor() after use.

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the device doesn't have a BOS descriptor
another LIBUSB_ERROR code on error

void API_EXPORTED libusb_free_bos_descriptor (struct libusb_bos_descriptor * *bos*)

Free a BOS descriptor obtained from libusb_get_bos_descriptor().

It is safe to call this function with a NULL bos parameter, in which case the function simply returns.

Parameters

bos	the BOS descriptor to free

int API_EXPORTED libusb_get_usb_2_0_extension_descriptor (struct libusb_context * *ctx,* struct libusb_bos_dev_capability_descriptor * *dev_cap,* struct libusb_usb_2_0_extension_descriptor ** *usb_2_0_extension*)

Get an USB 2.0 Extension descriptor.

Parameters

ctx	the context to operate on, or NULL for the default context
dev_cap	Device Capability descriptor with a bDevCapabilityType of libusb_capability_type::LIBUSB_BT_USB_2_0_EXTENSION LIBUSB_BT_USB_2_0_EXTENSION
usb_2_0_extension	output location for the USB 2.0 Extension descriptor. Only valid if 0 was returned. Must be freed with libusb_free_usb_2_0_extension_descriptor() after use.

Returns

0 on success
a LIBUSB_ERROR code on error

void API_EXPORTED libusb_free_usb_2_0_extension_descriptor (struct libusb_usb_2_0_extension_descriptor * *usb_2_0_extension*)

Free a USB 2.0 Extension descriptor obtained from libusb_get_usb_2_0_extension_descriptor().

It is safe to call this function with a NULL usb_2_0_extension parameter, in which case the function simply returns.

Parameters

usb_2_0_extension	the USB 2.0 Extension descriptor to free

int **API_EXPORTED** libusb_get_ss_usb_device_capability_descriptor (struct **libusb_context** ∗ *ctx,* struct **libusb_bos_dev_capability_descriptor** ∗ *dev_cap,* struct **libusb_ss_usb_device_capability_descriptor** ∗∗ *ss_usb_device_cap*)

Get a SuperSpeed USB Device Capability descriptor.

Parameters

ctx	the context to operate on, or NULL for the default context
dev_cap	Device Capability descriptor with a bDevCapabilityType of libusb_capability_type::LIBUSB_BT_SS_USB_DEVICE_CAPABILITY LIBUSB_BT_SS_USB_DEVICE_CAPABILITY
ss_usb_device_cap	output location for the SuperSpeed USB Device Capability descriptor. Only valid if 0 was returned. Must be freed with libusb_free_ss_usb_device_capability_descriptor() after use.

Returns

0 on success
a LIBUSB_ERROR code on error

void API_EXPORTED libusb_free_ss_usb_device_capability_descriptor (struct libusb_ss_usb_device_capability_descriptor * *ss_usb_device_cap*)

Free a SuperSpeed USB Device Capability descriptor obtained from libusb_get_ss_usb_device_capability_descriptor().

It is safe to call this function with a NULL ss_usb_device_cap parameter, in which case the function simply returns.

Parameters

ss_usb_device_cap	the USB 2.0 Extension descriptor to free

int API_EXPORTED libusb_get_container_id_descriptor (struct libusb_context * *ctx*, struct libusb_bos_dev_capability_descriptor * *dev_cap*, struct libusb_container_id_descriptor ** *container_id*)

Get a Container ID descriptor.

Parameters

ctx	the context to operate on, or NULL for the default context
dev_cap	Device Capability descriptor with a bDevCapabilityType of libusb_capability_type::LIBUSB_BT_CONTAINER_ID LIBUSB_BT_CONTAINER_ID
container_id	output location for the Container ID descriptor. Only valid if 0 was returned. Must be freed with libusb_free_container_id_descriptor() after use.

Returns

0 on success
a LIBUSB_ERROR code on error

void API_EXPORTED libusb_free_container_id_descriptor (struct libusb_container_id_descriptor * *container_id*)

Free a Container ID descriptor obtained from libusb_get_container_id_descriptor().

It is safe to call this function with a NULL container_id parameter, in which case the function simply returns.

Parameters

container_id	the USB 2.0 Extension descriptor to free

int API_EXPORTED libusb_get_string_descriptor_ascii (libusb_device_handle * *dev_handle*, uint8_t *desc_index*, unsigned char * *data*, int *length*)

Retrieve a string descriptor in C style ASCII.

Wrapper around libusb_get_string_descriptor(). Uses the first language supported by the device.

Parameters

dev_handle	a device handle
desc_index	the index of the descriptor to retrieve
data	output buffer for ASCII string descriptor
length	size of data buffer

Returns

number of bytes returned in data, or LIBUSB_ERROR code on failure

31.5 Device hotplug event notification

This page details how to use the libusb hotplug interface, where available.

Macros

- #define LIBUSB_HOTPLUG_MATCH_ANY -1

  Wildcard matching for hotplug events.

Typedefs

- typedef int libusb_hotplug_callback_handle

  Callback handle.
- typedef int(LIBUSB_CALL * libusb_hotplug_callback_fn)(libusb_context *ctx, libusb_device *device, libusb_hotplug_event event, void *user_data)

  Hotplug callback function type.

Enumerations

- enum libusb_hotplug_flag { LIBUSB_HOTPLUG_NO_FLAGS = 0, LIBUSB_HOTPLUG_ENUMERATE = 1<<0 }

  Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.
- enum libusb_hotplug_event { LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED = 0x01, LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT = 0x02 }

  Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.

Functions

- int LIBUSB_CALL libusb_hotplug_register_callback (libusb_context *ctx, libusb_hotplug_event events, libusb_hotplug_flag flags, int vendor_id, int product_id, int dev_class, libusb_hotplug_callback_fn cb_fn, void *user_data, libusb_hotplug_callback_handle *callback_handle)

  Register a hotplug callback function.
- void LIBUSB_CALL libusb_hotplug_deregister_callback (libusb_context *ctx, libusb_hotplug_callback_handle callback_handle)

  Deregisters a hotplug callback.

31.5.1 Detailed Description

This page details how to use the libusb hotplug interface, where available. Be mindful that not all platforms currently implement hotplug notification and that you should first call on libusb_has_capability() with parameter LIBUSB_CAP_HAS_HOTPLUG to confirm that hotplug support is available.

31.5.2 Macro Definition Documentation

#define LIBUSB_HOTPLUG_MATCH_ANY -1

Wildcard matching for hotplug events.

31.5.3 Typedef Documentation

typedef int libusb_hotplug_callback_handle

Callback handle.

Callbacks handles are generated by libusb_hotplug_register_callback() and can be used to deregister callbacks. Callback handles are unique per libusb_context and it is safe to call libusb_hotplug_deregister_callback() on an already deregisted callback.

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102

For more information, see Device hotplug event notification.

typedef int(LIBUSB_CALL * libusb_hotplug_callback_fn)(libusb_context *ctx, libusb_device *device, libusb_hotplug_event event, void *user_data)

Hotplug callback function type.

When requesting hotplug event notifications, you pass a pointer to a callback function of this type.

This callback may be called by an internal event thread and as such it is recommended the callback do minimal processing before returning.

libusb will call this function later, when a matching event had happened on a matching device. See Device hotplug event notification for more information.

It is safe to call either libusb_hotplug_register_callback() or libusb_hotplug_deregister_callback() from within a callback function.

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102

Parameters

ctx	context of this notification
device	libusb_device this event occurred on
event	event that occurred
user_data	user data provided when this callback was registered

Returns

bool whether this callback is finished processing events. returning 1 will cause this callback to be deregistered

31.5.4 Enumeration Type Documentation

enum libusb_hotplug_flag

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.

Flags for hotplug events

Enumerator

LIBUSB_HOTPLUG_NO_FLAGS Default value when not using any flags.

LIBUSB_HOTPLUG_ENUMERATE Arm the callback and fire it for all matching currently attached devices.

enum libusb_hotplug_event

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.

Hotplug events

Enumerator

LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED A device has been plugged in and is ready to use.

LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT A device has left and is no longer available. It is the user's responsibility to call libusb_close on any handle associated with a disconnected device. It is safe to call libusb_get_device_descriptor on a device that has left

31.5.5 Function Documentation

int LIBUSB_CALL libusb_hotplug_register_callback (libusb_context * *ctx*, libusb_hotplug_event *events*, libusb_hotplug_flag *flags*, int *vendor_id*, int *product_id*, int *dev_class*, libusb_hotplug_callback_fn *cb_fn*, void * *user_data*, libusb_hotplug_callback_handle * *callback_handle*)

Register a hotplug callback function.

Register a callback with the libusb_context. The callback will fire when a matching event occurs on a matching device. The callback is armed until either it is deregistered with libusb_hotplug_deregister_callback() or the supplied callback returns 1 to indicate it is finished processing events.

If the LIBUSB_HOTPLUG_ENUMERATE is passed the callback will be called with a LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED for all devices already plugged into the machine. Note that libusb modifies its internal device list from a separate thread, while calling hotplug callbacks from libusb_handle_events(), so it is possible for a device to already be present on, or removed from, its internal device list, while the hotplug callbacks still need to be dispatched. This means that when using LIBUSB_HOTPLUG_ENUMERATE, your callback may be called twice for the arrival of the same device, once from libusb_hotplug_register_callback() and once from libusb_handle_events(); and/or your callback may be called for the removal of a device for which an arrived call was never made.

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102

Parameters

in	*ctx*	context to register this callback with
in	*events*	bitwise or of events that will trigger this callback. See libusb_hotplug_event
in	*flags*	hotplug callback flags. See libusb_hotplug_flag
in	*vendor_id*	the vendor id to match or LIBUSB_HOTPLUG_MATCH_ANY
in	*product_id*	the product id to match or LIBUSB_HOTPLUG_MATCH_ANY
in	*dev_class*	the device class to match or LIBUSB_HOTPLUG_MATCH_ANY
in	*cb_fn*	the function to be invoked on a matching event/device
in	*user_data*	user data to pass to the callback function
out	*callback_handle*	pointer to store the handle of the allocated callback (can be NULL)

Returns

LIBUSB_SUCCESS on success LIBUSB_ERROR code on failure

void LIBUSB_CALL libusb_hotplug_deregister_callback (libusb_context * *ctx*, libusb_hotplug_callback_handle *callback_handle*)

Deregisters a hotplug callback.

Deregister a callback from a libusb_context. This function is safe to call from within a hotplug callback.

Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102

Parameters

in	*ctx*	context this callback is registered with
in	*callback_handle*	the handle of the callback to deregister

31.6 Asynchronous device I/O

This page details libusb's asynchronous (non-blocking) API for USB device I/O.

Data Structures

- struct libusb_control_setup

  Setup packet for control transfers.
- struct libusb_iso_packet_descriptor

  Isochronous packet descriptor.
- struct libusb_transfer

  The generic USB transfer structure.

Enumerations

- enum libusb_transfer_status { LIBUSB_TRANSFER_COMPLETED, LIBUSB_TRANSFER_ERROR, LIBUSB_TRANSFER_TIMED_OUT, LIBUSB_TRANSFER_CANCELLED, LIBUSB_TRANSFER_STALL, LIBUSB_TRANSFER_NO_DEVICE, LIBUSB_TRANSFER_OVERFLOW }

  Transfer status codes.
- enum libusb_transfer_flags { LIBUSB_TRANSFER_SHORT_NOT_OK = 1<<0, LIBUSB_TRANSFER_FREE_BUFFER = 1<<1, LIBUSB_TRANSFER_FREE_TRANSFER = 1<<2, LIBUSB_TRANSFER_ADD_ZERO_PACKET = 1 << 3 }

  libusb_transfer.flags values

Functions

- int API_EXPORTED libusb_alloc_streams (libusb_device_handle *dev_handle, uint32_t num_streams, unsigned char *endpoints, int num_endpoints)

  Allocate up to num_streams usb bulk streams on the specified endpoints.
- int API_EXPORTED libusb_free_streams (libusb_device_handle *dev_handle, unsigned char *endpoints, int num_endpoints)

  Free usb bulk streams allocated with libusb_alloc_streams().
- DEFAULT_VISIBILITY unsigned char *LIBUSB_CALL libusb_dev_mem_alloc (libusb_device_handle *dev_handle, size_t length)

  Attempts to allocate a block of persistent DMA memory suitable for transfers against the given device.
- int API_EXPORTED libusb_dev_mem_free (libusb_device_handle *dev_handle, unsigned char *buffer, size_t length)

  Free device memory allocated with libusb_dev_mem_alloc().
- DEFAULT_VISIBILITY struct libusb_transfer *LIBUSB_CALL libusb_alloc_transfer (int iso_packets)

  Allocate a libusb transfer with a specified number of isochronous packet descriptors.
- void API_EXPORTED libusb_free_transfer (struct libusb_transfer *transfer)

  Free a transfer structure.
- int API_EXPORTED libusb_submit_transfer (struct libusb_transfer *transfer)

  Submit a transfer.
- int API_EXPORTED libusb_cancel_transfer (struct libusb_transfer *transfer)

  Asynchronously cancel a previously submitted transfer.
- void API_EXPORTED libusb_transfer_set_stream_id (struct libusb_transfer *transfer, uint32_t stream_id)

  Set a transfers bulk stream id.
- uint32_t API_EXPORTED libusb_transfer_get_stream_id (struct libusb_transfer *transfer)

  Get a transfers bulk stream id.
- typedef void (LIBUSB_CALL *libusb_transfer_cb_fn)(struct libusb_transfer *transfer)

  Asynchronous transfer callback function type.

31.6.1 Detailed Description

This page details libusb's asynchronous (non-blocking) API for USB device I/O. This interface is very powerful but is also quite complex - you will need to read this page carefully to understand the necessary considerations and issues surrounding use of this interface. Simplistic applications may wish to consider the synchronous I/O API instead.

The asynchronous interface is built around the idea of separating transfer submission and handling of transfer completion (the synchronous model combines both of these into one). There may be a long delay between submission and completion, however the asynchronous submission function is non-blocking so will return control to your application during that potentially long delay.

31.6.2 Transfer abstraction

For the asynchronous I/O, libusb implements the concept of a generic transfer entity for all types of I/O (control, bulk, interrupt, isochronous). The generic transfer object must be treated slightly differently depending on which type of I/O you are performing with it.

This is represented by the public libusb_transfer structure type.

31.6.3 Asynchronous transfers

We can view asynchronous I/O as a 5 step process:

1. **Allocation**: allocate a libusb_transfer
2. **Filling**: populate the libusb_transfer instance with information about the transfer you wish to perform
3. **Submission**: ask libusb to submit the transfer
4. **Completion handling**: examine transfer results in the libusb_transfer structure
5. **Deallocation**: clean up resources

Allocation

This step involves allocating memory for a USB transfer. This is the generic transfer object mentioned above. At this stage, the transfer is "blank" with no details about what type of I/O it will be used for.

Allocation is done with the libusb_alloc_transfer() function. You must use this function rather than allocating your own transfers.

Filling

This step is where you take a previously allocated transfer and fill it with information to determine the message type and direction, data buffer, callback function, etc.

You can either fill the required fields yourself or you can use the helper functions: libusb_fill_control_transfer(), libusb_fill_bulk_transfer() and libusb_fill_interrupt_transfer().

Submission

When you have allocated a transfer and filled it, you can submit it using libusb_submit_transfer(). This function returns immediately but can be regarded as firing off the I/O request in the background.

Completion handling

After a transfer has been submitted, one of four things can happen to it:

- The transfer completes (i.e. some data was transferred)
- The transfer has a timeout and the timeout expires before all data is transferred
- The transfer fails due to an error
- The transfer is cancelled

Each of these will cause the user-specified transfer callback function to be invoked. It is up to the callback function to determine which of the above actually happened and to act accordingly.

The user-specified callback is passed a pointer to the libusb_transfer structure which was used to setup and submit the transfer. At completion time, libusb has populated this structure with results of the transfer: success or failure reason, number of bytes of data transferred, etc. See the libusb_transfer structure documentation for more information.

Important Note: The user-specified callback is called from an event handling context. It is therefore important that no calls are made into libusb that will attempt to perform any event handling. Examples of such functions are any listed in the synchronous API and any of the blocking functions that retrieve USB descriptors.

Deallocation

When a transfer has completed (i.e. the callback function has been invoked), you are advised to free the transfer (unless you wish to resubmit it, see below). Transfers are deallocated with libusb_free_transfer().

It is undefined behaviour to free a transfer which has not completed.

31.6.4 Resubmission

You may be wondering why allocation, filling, and submission are all separated above where they could reasonably be combined into a single operation.

The reason for separation is to allow you to resubmit transfers without having to allocate new ones every time. This is especially useful for common situations dealing with interrupt endpoints - you allocate one transfer, fill and submit it, and when it returns with results you just resubmit it for the next interrupt.

31.6.5 Cancellation

Another advantage of using the asynchronous interface is that you have the ability to cancel transfers which have not yet completed. This is done by calling the libusb_cancel_transfer() function.

libusb_cancel_transfer() is asynchronous/non-blocking in itself. When the cancellation actually completes, the transfer's callback function will be invoked, and the callback function should check the transfer status to determine that it was cancelled.

Freeing the transfer after it has been cancelled but before cancellation has completed will result in undefined behaviour.

When a transfer is cancelled, some of the data may have been transferred. libusb will communicate this to you in the transfer callback. Do not assume that no data was transferred.

31.6.6 Overflows on device-to-host bulk/interrupt endpoints

If your device does not have predictable transfer sizes (or it misbehaves), your application may submit a request for data on an IN endpoint which is smaller than the data that the device wishes to send. In some circumstances this will cause an overflow, which is a nasty condition to deal with. See the Packets and overflows page for discussion.

31.6.7 Considerations for control transfers

The `libusb_transfer` structure is generic and hence does not include specific fields for the control-specific setup packet structure.

In order to perform a control transfer, you must place the 8-byte setup packet at the start of the data buffer. To simplify this, you could cast the buffer pointer to type struct libusb_control_setup, or you can use the helper function libusb_fill_control_setup().

The wLength field placed in the setup packet must be the length you would expect to be sent in the setup packet: the length of the payload that follows (or the expected maximum number of bytes to receive). However, the length field of the libusb_transfer object must be the length of the data buffer - i.e. it should be wLength *plus* the size of the setup packet (LIBUSB_CONTROL_SETUP_SIZE).

If you use the helper functions, this is simplified for you:

1. Allocate a buffer of size LIBUSB_CONTROL_SETUP_SIZE plus the size of the data you are sending/requesting.
2. Call libusb_fill_control_setup() on the data buffer, using the transfer request size as the wLength value (i.e. do not include the extra space you allocated for the control setup).
3. If this is a host-to-device transfer, place the data to be transferred in the data buffer, starting at offset LIBUSB_CONTROL_SETUP_SIZE.
4. Call libusb_fill_control_transfer() to associate the data buffer with the transfer (and to set the remaining details such as callback and timeout).
   - Note that there is no parameter to set the length field of the transfer. The length is automatically inferred from the wLength field of the setup packet.
5. Submit the transfer.

The multi-byte control setup fields (wValue, wIndex and wLength) must be given in little-endian byte order (the endianness of the USB bus). Endianness conversion is transparently handled by libusb_fill_control_setup() which is documented to accept host-endian values.

Further considerations are needed when handling transfer completion in your callback function:

- As you might expect, the setup packet will still be sitting at the start of the data buffer.
- If this was a device-to-host transfer, the received data will be sitting at offset LIBUSB_CONTROL_SETUP_SIZE into the buffer.
- The actual_length field of the transfer structure is relative to the wLength of the setup packet, rather than the size of the data buffer. So, if your wLength was 4, your transfer's `length` was 12, then you should expect an `actual_length` of 4 to indicate that the data was transferred in entirity.

To simplify parsing of setup packets and obtaining the data from the correct offset, you may wish to use the libusb_control_transfer_get_data() and libusb_control_transfer_get_setup() functions within your transfer callback.

Even though control endpoints do not halt, a completed control transfer may have a LIBUSB_TRANSFER_STALL status code. This indicates the control request was not supported.

31.6.8 Considerations for interrupt transfers

All interrupt transfers are performed using the polling interval presented by the bInterval value of the endpoint descriptor.

31.6.9 Considerations for isochronous transfers

Isochronous transfers are more complicated than transfers to non-isochronous endpoints.

To perform I/O to an isochronous endpoint, allocate the transfer by calling libusb_alloc_transfer() with an appropriate number of isochronous packets.

During filling, set type to LIBUSB_TRANSFER_TYPE_ISOCHRONOUS, and set num_iso_packets to a value less than or equal to the number of packets you requested during allocation. libusb_alloc_transfer() does not set either of these fields for you, given that you might not even use the transfer on an isochronous endpoint.

Next, populate the length field for the first num_iso_packets entries in the iso_packet_desc array. Section 5.6.3 of the USB2 specifications describe how the maximum isochronous packet length is determined by the wMaxPacketSize field in the endpoint descriptor. Two functions can help you here:

- libusb_get_max_iso_packet_size() is an easy way to determine the max packet size for an isochronous endpoint. Note that the maximum packet size is actually the maximum number of bytes that can be transmitted in a single microframe, therefore this function multiplies the maximum number of bytes per transaction by the number of transaction opportunities per microframe.
- libusb_set_iso_packet_lengths() assigns the same length to all packets within a transfer, which is usually what you want.

For outgoing transfers, you'll obviously fill the buffer and populate the packet descriptors in hope that all the data gets transferred. For incoming transfers, you must ensure the buffer has sufficient capacity for the situation where all packets transfer the full amount of requested data.

Completion handling requires some extra consideration. The actual_length field of the transfer is meaningless and should not be examined; instead you must refer to the actual_length field of each individual packet.

The status field of the transfer is also a little misleading:

- If the packets were submitted and the isochronous data microframes completed normally, status will have value LIBUSB_TRANSFER_COMPLETED. Note that bus errors and software-incurred delays are not counted as transfer errors; the transfer.status field may indicate COMPLETED even if some or all of the packets failed. Refer to the status field of each individual packet to determine packet failures.
- The status field will have value LIBUSB_TRANSFER_ERROR only when serious errors were encountered.
- Other transfer status codes occur with normal behaviour.

The data for each packet will be found at an offset into the buffer that can be calculated as if each prior packet completed in full. The libusb_get_iso_packet_buffer() and libusb_get_iso_packet_buffer_simple() functions may help you here.

Note: Some operating systems (e.g. Linux) may impose limits on the length of individual isochronous packets and/or the total length of the isochronous transfer. Such limits can be difficult for libusb to detect, so the library will simply try and submit the transfer as set up by you. If the transfer fails to submit because it is too large, libusb_submit_transfer() will return LIBUSB_ERROR_INVALID_PARAM.

31.6.10 Memory caveats

In most circumstances, it is not safe to use stack memory for transfer buffers. This is because the function that fired off the asynchronous transfer may return before libusb has finished using the buffer, and when the function returns it's stack gets destroyed. This is true for both host-to-device and device-to-host transfers.

The only case in which it is safe to use stack memory is where you can guarantee that the function owning the stack space for the buffer does not return until after the transfer's callback function has completed. In every other case, you need to use heap memory instead.

31.6.11 Fine control

Through using this asynchronous interface, you may find yourself repeating a few simple operations many times. You can apply a bitwise OR of certain flags to a transfer to simplify certain things:

- LIBUSB_TRANSFER_SHORT_NOT_OK results in transfers which transferred less than the requested amount of data being marked with status LIBUSB_TRANSFER_ERROR (they would normally be regarded as COMPLETED)
- LIBUSB_TRANSFER_FREE_BUFFER allows you to ask libusb to free the transfer buffer when freeing the transfer.
- LIBUSB_TRANSFER_FREE_TRANSFER causes libusb to automatically free the transfer after the transfer callback returns.

31.6.12 Event handling

An asynchronous model requires that libusb perform work at various points in time - namely processing the results of previously-submitted transfers and invoking the user-supplied callback function.

This gives rise to the libusb_handle_events() function which your application must call into when libusb has work do to. This gives libusb the opportunity to reap pending transfers, invoke callbacks, etc.

There are 2 different approaches to dealing with libusb_handle_events:

1. Repeatedly call libusb_handle_events() in blocking mode from a dedicated thread.
2. Integrate libusb with your application's main event loop. libusb exposes a set of file descriptors which allow you to do this.

The first approach has the big advantage that it will also work on Windows were libusb' poll API for select / poll integration is not available. So if you want to support Windows and use the async API, you must use this approach, see the Using an event handling thread section below for details.

If you prefer a single threaded approach with a single central event loop, see the polling and timing section for how to integrate libusb into your application's main event loop.

31.6.13 Using an event handling thread

Lets begin with stating the obvious: If you're going to use a separate thread for libusb event handling, your callback functions MUST be threadsafe.

Other then that doing event handling from a separate thread, is mostly simple. You can use an event thread function as follows:

```
void *event_thread_func(void *ctx)
{
    while (event_thread_run)
        libusb_handle_events(ctx);

    return NULL;
}
```

There is one caveat though, stopping this thread requires setting the event_thread_run variable to 0, and after that libusb_handle_events() needs to return control to event_thread_func. But unless some event happens, libusb_handle_events() will not return.

There are 2 different ways of dealing with this, depending on if your application uses libusb' hotplug support or not.

Applications which do not use hotplug support, should not start the event thread until after their first call to libusb_open(), and should stop the thread when closing the last open device as follows:

```
void my_close_handle(libusb_device_handle *dev_handle)
{
    if (open_devs == 1)
        event_thread_run = 0;

    libusb_close(dev_handle); // This wakes up libusb_handle_events()

    if (open_devs == 1)
        pthread_join(event_thread);

    open_devs--;
}
```

Applications using hotplug support should start the thread at program init, after having successfully called libusb_hotplug_register_callback(), and should stop the thread at program exit as follows:

```
void my_libusb_exit(void)
{
    event_thread_run = 0;
    libusb_hotplug_deregister_callback(ctx, hotplug_cb_handle); // This
       wakes up libusb_handle_events()
    pthread_join(event_thread);
    libusb_exit(ctx);
}
```

31.6.14 Enumeration Type Documentation

enum libusb_transfer_status

Transfer status codes.

Enumerator

LIBUSB_TRANSFER_COMPLETED Transfer completed without error. Note that this does not indicate that the entire amount of requested data was transferred.

LIBUSB_TRANSFER_ERROR Transfer failed.

LIBUSB_TRANSFER_TIMED_OUT Transfer timed out.

LIBUSB_TRANSFER_CANCELLED Transfer was cancelled.

LIBUSB_TRANSFER_STALL For bulk/interrupt endpoints: halt condition detected (endpoint stalled). For control endpoints: control request not supported.

LIBUSB_TRANSFER_NO_DEVICE Device was disconnected.

LIBUSB_TRANSFER_OVERFLOW Device sent more data than requested.

enum libusb_transfer_flags

libusb_transfer.flags values

Enumerator

LIBUSB_TRANSFER_SHORT_NOT_OK Report short frames as errors.

LIBUSB_TRANSFER_FREE_BUFFER Automatically free() transfer buffer during libusb_free_transfer(). Note that buffers allocated with libusb_dev_mem_alloc() should not be attempted freed in this way, since free() is not an appropriate way to release such memory.

LIBUSB_TRANSFER_FREE_TRANSFER Automatically call libusb_free_transfer() after callback returns. If this flag is set, it is illegal to call libusb_free_transfer() from your transfer callback, as this will result in a double-free when this flag is acted upon.

LIBUSB_TRANSFER_ADD_ZERO_PACKET Terminate transfers that are a multiple of the endpoint's wMaxPacketSize with an extra zero length packet. This is useful when a device protocol mandates that each logical request is terminated by an incomplete packet (i.e. the logical requests are not separated by other means).

This flag only affects host-to-device transfers to bulk and interrupt endpoints. In other situations, it is ignored.

This flag only affects transfers with a length that is a multiple of the endpoint's wMaxPacketSize. On transfers of other lengths, this flag has no effect. Therefore, if you are working with a device that needs a ZLP whenever the end of the logical request falls on a packet boundary, then it is sensible to set this flag on *every* transfer (you do not have to worry about only setting it on transfers that end on the boundary).

This flag is currently only supported on Linux. On other systems, libusb_submit_transfer() will return LIBUSB_ERROR_NOT_SUPPORTED for every transfer where this flag is set.

Available since libusb-1.0.9.

31.6.15 Function Documentation

int API_EXPORTED libusb_alloc_streams (libusb_device_handle * *dev_handle*, uint32_t *num_streams*, unsigned char * *endpoints*, int *num_endpoints*)

Allocate up to num_streams usb bulk streams on the specified endpoints.

This function takes an array of endpoints rather then a single endpoint because some protocols require that endpoints are setup with similar stream ids. All endpoints passed in must belong to the same interface.

Note this function may return less streams then requested. Also note that the same number of streams are allocated for each endpoint in the endpoint array.

Stream id 0 is reserved, and should not be used to communicate with devices. If libusb_alloc_streams() returns with a value of N, you may use stream ids 1 to N.

Since version 1.0.19, LIBUSB_API_VERSION >= 0x01000103

Parameters

dev_handle	a device handle
num_streams	number of streams to try to allocate

endpoints	array of endpoints to allocate streams on
num_endpoints	length of the endpoints array

Returns

number of streams allocated, or a LIBUSB_ERROR code on failure

int API_EXPORTED libusb_free_streams (libusb_device_handle * *dev_handle,* unsigned char * *endpoints,* int *num_endpoints*)

Free usb bulk streams allocated with libusb_alloc_streams().

Note streams are automatically free-ed when releasing an interface.

Since version 1.0.19, LIBUSB_API_VERSION >= 0x01000103

Parameters

dev_handle	a device handle
endpoints	array of endpoints to free streams on
num_endpoints	length of the endpoints array

Returns

LIBUSB_SUCCESS, or a LIBUSB_ERROR code on failure

DEFAULT_VISIBILITY unsigned char* LIBUSB_CALL libusb_dev_mem_alloc (libusb_device_handle * *dev_handle,* size_t *length*)

Attempts to allocate a block of persistent DMA memory suitable for transfers against the given device.

If successful, will return a block of memory that is suitable for use as "buffer" in libusb_transfer against this device. Using this memory instead of regular memory means that the host controller can use DMA directly into the buffer to increase performance, and also that transfers can no longer fail due to kernel memory fragmentation.

Note that this means you should not modify this memory (or even data on the same cache lines) when a transfer is in progress, although it is legal to have several transfers going on within the same memory block.

Will return NULL on failure. Many systems do not support such zerocopy and will always return NULL. Memory allocated with this function must be freed with libusb_dev_mem_free. Specifically, this means that the flag LIBUSB_TRANSFER_FREE_BUFFER cannot be used to free memory allocated with this function.

Since version 1.0.21, LIBUSB_API_VERSION >= 0x01000105

Parameters

dev_handle	a device handle
length	size of desired data buffer

Returns

a pointer to the newly allocated memory, or NULL on failure

int API_EXPORTED libusb_dev_mem_free (libusb_device_handle * *dev_handle,* unsigned char * *buffer,* size_t *length*)

Free device memory allocated with libusb_dev_mem_alloc().

Parameters

dev_handle	a device handle
buffer	pointer to the previously allocated memory
length	size of previously allocated memory

Returns

LIBUSB_SUCCESS, or a LIBUSB_ERROR code on failure

DEFAULT_VISIBILITY struct libusb_transfer* LIBUSB_CALL libusb_alloc_transfer (int *iso_packets*)

Allocate a libusb transfer with a specified number of isochronous packet descriptors.

The returned transfer is pre-initialized for you. When the new transfer is no longer needed, it should be freed with libusb_free_transfer().

Transfers intended for non-isochronous endpoints (e.g. control, bulk, interrupt) should specify an iso_packets count of zero.

For transfers intended for isochronous endpoints, specify an appropriate number of packet descriptors to be allocated as part of the transfer. The returned transfer is not specially initialized for isochronous I/O; you are still required to set the num_iso_packets and type fields accordingly.

It is safe to allocate a transfer with some isochronous packets and then use it on a non-isochronous endpoint. If you do this, ensure that at time of submission, num_iso_packets is 0 and that type is set appropriately.

Parameters

iso_packets	number of isochronous packet descriptors to allocate

Returns

a newly allocated transfer, or NULL on error

void API_EXPORTED libusb_free_transfer (struct libusb_transfer * *transfer*)

Free a transfer structure.

This should be called for all transfers allocated with libusb_alloc_transfer().

If the LIBUSB_TRANSFER_FREE_BUFFER flag is set and the transfer buffer is non-NULL, this function will also free the transfer buffer using the standard system memory allocator (e.g. free()).

It is legal to call this function with a NULL transfer. In this case, the function will simply return safely.

It is not legal to free an active transfer (one which has been submitted and has not yet completed).

Parameters

transfer	the transfer to free

int API_EXPORTED libusb_submit_transfer (struct libusb_transfer * *transfer*)

Submit a transfer.

This function will fire off the USB transfer and then return immediately.

Parameters

transfer	the transfer to submit

Returns

0 on success
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_BUSY if the transfer has already been submitted.
LIBUSB_ERROR_NOT_SUPPORTED if the transfer flags are not supported by the operating system.
LIBUSB_ERROR_INVALID_PARAM if the transfer size is larger than the operating system and/or hardware can support
another LIBUSB_ERROR code on other failure

int API_EXPORTED libusb_cancel_transfer (struct libusb_transfer * *transfer*)

Asynchronously cancel a previously submitted transfer.

This function returns immediately, but this does not indicate cancellation is complete. Your callback function will be invoked at some later time with a transfer status of LIBUSB_TRANSFER_CANCELLED.

Parameters

transfer	the transfer to cancel

Returns

0 on success
LIBUSB_ERROR_NOT_FOUND if the transfer is not in progress, already complete, or already cancelled.
a LIBUSB_ERROR code on failure

void API_EXPORTED libusb_transfer_set_stream_id (struct libusb_transfer * *transfer,* uint32_t *stream_id*)

Set a transfers bulk stream id.

Note users are advised to use libusb_fill_bulk_stream_transfer() instead of calling this function directly.

Since version 1.0.19, LIBUSB_API_VERSION >= 0x01000103

Parameters

transfer	the transfer to set the stream id for
stream_id	the stream id to set

See Also

libusb_alloc_streams()

uint32_t API_EXPORTED libusb_transfer_get_stream_id (struct libusb_transfer * *transfer*)

Get a transfers bulk stream id.

Since version 1.0.19, LIBUSB_API_VERSION >= 0x01000103

Parameters

transfer	the transfer to get the stream id for

Returns

the stream id for the transfer

typedef void (LIBUSB_CALL * *libusb_transfer_cb_fn*)

Asynchronous transfer callback function type.

When submitting asynchronous transfers, you pass a pointer to a callback function of this type via the callback member of the libusb_transfer structure. libusb will call this function later, when the transfer has completed or failed. See Asynchronous device I/O for more information.

Parameters

transfer	The libusb_transfer struct the callback function is being notified about.

31.7 Polling and timing

This page documents libusb's functions for polling events and timing.

Data Structures

- struct libusb_pollfd

  File descriptor for polling.

Typedefs

- typedef void(LIBUSB_CALL * libusb_pollfd_added_cb)(int fd, short events, void *user_data)

  Callback function, invoked when a new file descriptor should be added to the set of file descriptors monitored for events.
- typedef void(LIBUSB_CALL * libusb_pollfd_removed_cb)(int fd, void *user_data)

  Callback function, invoked when a file descriptor should be removed from the set of file descriptors being monitored for events.

Functions

- int API_EXPORTED libusb_try_lock_events (libusb_context *ctx)

  Attempt to acquire the event handling lock.
- void API_EXPORTED libusb_lock_events (libusb_context *ctx)

  Acquire the event handling lock, blocking until successful acquisition if it is contended.
- void API_EXPORTED libusb_unlock_events (libusb_context *ctx)

  Release the lock previously acquired with libusb_try_lock_events() or libusb_lock_events().
- int API_EXPORTED libusb_event_handling_ok (libusb_context *ctx)

  Determine if it is still OK for this thread to be doing event handling.
- int API_EXPORTED libusb_event_handler_active (libusb_context *ctx)

  Determine if an active thread is handling events (i.e.
- void API_EXPORTED libusb_interrupt_event_handler (libusb_context *ctx)

  Interrupt any active thread that is handling events.
- void API_EXPORTED libusb_lock_event_waiters (libusb_context *ctx)

  Acquire the event waiters lock.
- void API_EXPORTED libusb_unlock_event_waiters (libusb_context *ctx)

  Release the event waiters lock.
- int API_EXPORTED libusb_wait_for_event (libusb_context *ctx, struct timeval *tv)

  Wait for another thread to signal completion of an event.
- int API_EXPORTED libusb_handle_events_timeout_completed (libusb_context *ctx, struct timeval *tv, int *completed)

  Handle any pending events.
- int API_EXPORTED libusb_handle_events_timeout (libusb_context *ctx, struct timeval *tv)

  Handle any pending events.
- int API_EXPORTED libusb_handle_events (libusb_context *ctx)

  Handle any pending events in blocking mode.
- int API_EXPORTED libusb_handle_events_completed (libusb_context *ctx, int *completed)

  Handle any pending events in blocking mode.
- int API_EXPORTED libusb_handle_events_locked (libusb_context *ctx, struct timeval *tv)

  Handle any pending events by polling file descriptors, without checking if any other threads are already doing so.
- int API_EXPORTED libusb_pollfds_handle_timeouts (libusb_context *ctx)

  Determines whether your application must apply special timing considerations when monitoring libusb's file descriptors.
- int API_EXPORTED libusb_get_next_timeout (libusb_context *ctx, struct timeval *tv)

  Determine the next internal timeout that libusb needs to handle.
- void API_EXPORTED libusb_set_pollfd_notifiers (libusb_context *ctx, libusb_pollfd_added_cb added_cb, libusb_pollfd_removed_cb removed_cb, void *user_data)

  Register notification functions for file descriptor additions/removals.
- DEFAULT_VISIBILITY const struct libusb_pollfd **LIBUSB_CALL libusb_get_pollfds (libusb_context *ctx)

  Retrieve a list of file descriptors that should be polled by your main loop as libusb event sources.
- void API_EXPORTED libusb_free_pollfds (const struct libusb_pollfd **pollfds)

  Free a list of libusb_pollfd structures.

31.7.1 Detailed Description

This page documents libusb's functions for polling events and timing. These functions are only necessary for users of the asynchronous API. If you are only using the simpler synchronous API then you do not need to ever call these functions.

The justification for the functionality described here has already been discussed in the event handling section of the asynchronous API documentation. In summary, libusb does not create internal threads for event processing and hence relies on your application calling into libusb at certain points in time so that pending events can be handled.

Your main loop is probably already calling poll() or select() or a variant on a set of file descriptors for other event sources (e.g. keyboard button presses, mouse movements, network sockets, etc). You then add libusb's file descriptors to your poll()/select() calls, and when activity is detected on such descriptors you know it is time to call libusb_handle_events().

There is one final event handling complication. libusb supports asynchronous transfers which time out after a specified time period.

On some platforms a timerfd is used, so the timeout handling is just another fd, on other platforms this requires that libusb is called into at or after the timeout to handle it. So, in addition to considering libusb's file descriptors in your main event loop, you must also consider that libusb sometimes needs to be called into at fixed points in time even when there is no file descriptor activity, see Notes on time-based events details.

In order to know precisely when libusb needs to be called into, libusb offers you a set of pollable file descriptors and information about when the next timeout expires.

If you are using the asynchronous I/O API, you must take one of the two following options, otherwise your I/O will not complete.

31.7.2 The simple option

If your application revolves solely around libusb and does not need to handle other event sources, you can have a program structure as follows:

```
// initialize libusb
// find and open device
// maybe fire off some initial async I/O

while (user_has_not_requested_exit)
        libusb_handle_events(ctx);

// clean up and exit
```

With such a simple main loop, you do not have to worry about managing sets of file descriptors or handling timeouts. libusb_handle_events() will handle those details internally.

31.7.3 The more advanced option

Note

> This functionality is currently only available on Unix-like platforms. On Windows, libusb_get_pollfds() simply returns NULL. Applications which want to support Windows are advised to use an event handling thread instead.

In more advanced applications, you will already have a main loop which is monitoring other event sources: network sockets, X11 events, mouse movements, etc. Through exposing a set of file descriptors, libusb is designed to cleanly integrate into such main loops.

In addition to polling file descriptors for the other event sources, you take a set of file descriptors from libusb and monitor those too. When you detect activity on libusb's file descriptors, you call libusb_handle_events_timeout() in non-blocking mode.

What's more, libusb may also need to handle events at specific moments in time. No file descriptor activity is generated at these times, so your own application needs to be continually aware of when the next one of these moments occurs (through calling libusb_get_next_timeout()), and then it needs to call libusb_handle_events_timeout() in non-blocking mode when these moments occur. This means that you need to adjust your poll()/select() timeout accordingly.

libusb provides you with a set of file descriptors to poll and expects you to poll all of them, treating them as a single entity. The meaning of each file descriptor in the set is an internal implementation detail, platform-dependent and may vary from release to release. Don't try and interpret the meaning of the file descriptors, just do as libusb indicates, polling all of them at once.

In pseudo-code, you want something that looks like:

```
// initialise libusb

libusb_get_pollfds(ctx)
while (user has not requested application exit) {
        libusb_get_next_timeout(ctx);
        poll(on libusb file descriptors plus any other event sources of interest,
                using a timeout no larger than the value libusb just suggested)
        if (poll() indicated activity on libusb file descriptors)
                libusb_handle_events_timeout(ctx, &zero_tv);
```

```
        if (time has elapsed to or beyond the libusb timeout)
                libusb_handle_events_timeout(ctx, &zero_tv);
        // handle events from other sources here
}

// clean up and exit
```

Notes on time-based events

The above complication with having to track time and call into libusb at specific moments is a bit of a headache. For maximum compatibility, you do need to write your main loop as above, but you may decide that you can restrict the supported platforms of your application and get away with a more simplistic scheme.

These time-based event complications are **not** required on the following platforms:

- Darwin
- Linux, provided that the following version requirements are satisfied:
  - Linux v2.6.27 or newer, compiled with timerfd support
  - glibc v2.9 or newer
  - libusb v1.0.5 or newer

Under these configurations, libusb_get_next_timeout() will *always* return 0, so your main loop can be simplified to:

```
// initialise libusb

libusb_get_pollfds(ctx)
while (user has not requested application exit) {
        poll(on libusb file descriptors plus any other event sources of interest,
                using any timeout that you like)
        if (poll() indicated activity on libusb file descriptors)
                libusb_handle_events_timeout(ctx, &zero_tv);
        // handle events from other sources here
}

// clean up and exit
```

Do remember that if you simplify your main loop to the above, you will lose compatibility with some platforms (including legacy Linux platforms, and *any future platforms supported by libusb which may have time-based event requirements*). The resultant problems will likely appear as strange bugs in your application.

You can use the libusb_pollfds_handle_timeouts() function to do a runtime check to see if it is safe to ignore the time-based event complications. If your application has taken the shortcut of ignoring libusb's next timeout in your main loop, then you are advised to check the return value of libusb_pollfds_handle_timeouts() during application startup, and to abort if the platform does suffer from these timing complications.

Changes in the file descriptor set

The set of file descriptors that libusb uses as event sources may change during the life of your application. Rather than having to repeatedly call libusb_get_pollfds(), you can set up notification functions for when the file descriptor set changes using libusb_set_pollfd_notifiers().

Multi-threaded considerations

Unfortunately, the situation is complicated further when multiple threads come into play. If two threads are monitoring the same file descriptors, the fact that only one thread will be woken up when an event occurs causes some headaches.

The events lock, event waiters lock, and libusb_handle_events_locked() entities are added to solve these problems. You do not need to be concerned with these entities otherwise.

See the extra documentation: Multi-threaded applications and asynchronous I/O

31.7.4 Typedef Documentation

typedef void(LIBUSB_CALL ∗ libusb_pollfd_added_cb)(int fd, short events, void ∗user_data)

Callback function, invoked when a new file descriptor should be added to the set of file descriptors monitored for events.

Parameters

fd	the new file descriptor

events	events to monitor for, see libusb_pollfd for a description
user_data	User data pointer specified in libusb_set_pollfd_notifiers() call

See Also

libusb_set_pollfd_notifiers()

typedef void(LIBUSB_CALL * libusb_pollfd_removed_cb)(int fd, void *user_data)

Callback function, invoked when a file descriptor should be removed from the set of file descriptors being monitored for events.

After returning from this callback, do not use that file descriptor again.

Parameters

fd	the file descriptor to stop monitoring
user_data	User data pointer specified in libusb_set_pollfd_notifiers() call

See Also

libusb_set_pollfd_notifiers()

31.7.5 Function Documentation

int API_EXPORTED libusb_try_lock_events (libusb_context * *ctx*)

Attempt to acquire the event handling lock.

This lock is used to ensure that only one thread is monitoring libusb event sources at any one time.

You only need to use this lock if you are developing an application which calls poll() or select() on libusb's file descriptors directly. If you stick to libusb's event handling loop functions (e.g. libusb_handle_events()) then you do not need to be concerned with this locking.

While holding this lock, you are trusted to actually be handling events. If you are no longer handling events, you must call libusb_unlock_events() as soon as possible.

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

0 if the lock was obtained successfully
1 if the lock was not obtained (i.e. another thread holds the lock) Multi-threaded applications and asynchronous I/O

void API_EXPORTED libusb_lock_events (libusb_context * *ctx*)

Acquire the event handling lock, blocking until successful acquisition if it is contended.

This lock is used to ensure that only one thread is monitoring libusb event sources at any one time.

You only need to use this lock if you are developing an application which calls poll() or select() on libusb's file descriptors directly. If you stick to libusb's event handling loop functions (e.g. libusb_handle_events()) then you do not need to be concerned with this locking.

While holding this lock, you are trusted to actually be handling events. If you are no longer handling events, you must call libusb_unlock_events() as soon as possible.

Parameters

ctx	the context to operate on, or NULL for the default context Multi-threaded applications and asynchronous I/O

void API_EXPORTED libusb_unlock_events (libusb_context * *ctx*)

Release the lock previously acquired with libusb_try_lock_events() or libusb_lock_events().

Releasing this lock will wake up any threads blocked on libusb_wait_for_event().

Parameters

ctx	the context to operate on, or NULL for the default context Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_event_handling_ok (libusb_context * *ctx*)

Determine if it is still OK for this thread to be doing event handling.

Sometimes, libusb needs to temporarily pause all event handlers, and this is the function you should use before polling file descriptors to see if this is the case.

If this function instructs your thread to give up the events lock, you should just continue the usual logic that is documented in Multi-threaded applications and asynchronous I/O. On the next iteration, your thread will fail to obtain the events lock, and will hence become an event waiter.

This function should be called while the events lock is held: you don't need to worry about the results of this function if your thread is not the current event handler.

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

1 if event handling can start or continue
0 if this thread must give up the events lock Multi-threaded I/O: the full story

int API_EXPORTED libusb_event_handler_active (libusb_context * *ctx*)

Determine if an active thread is handling events (i.e.

if anyone is holding the event handling lock).

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

1 if a thread is handling events
0 if there are no threads currently handling events Multi-threaded applications and asynchronous I/O

void API_EXPORTED libusb_interrupt_event_handler (libusb_context * *ctx*)

Interrupt any active thread that is handling events.

This is mainly useful for interrupting a dedicated event handling thread when an application wishes to call libusb_exit().

Since version 1.0.21, LIBUSB_API_VERSION >= 0x01000105

Parameters

ctx	the context to operate on, or NULL for the default context Multi-threaded applications and asynchronous I/O

void API_EXPORTED libusb_lock_event_waiters (libusb_context * *ctx*)

Acquire the event waiters lock.

This lock is designed to be obtained under the situation where you want to be aware when events are completed, but some other thread is event handling so calling libusb_handle_events() is not allowed.

You then obtain this lock, re-check that another thread is still handling events, then call libusb_wait_for_event().

You only need to use this lock if you are developing an application which calls poll() or select() on libusb's file descriptors directly, **and** may potentially be handling events from 2 threads simultaenously. If you stick to libusb's event handling loop functions (e.g. libusb_handle_events()) then you do not need to be concerned with this locking.

Parameters

ctx	the context to operate on, or NULL for the default context Multi-threaded applications and asynchronous I/O

void API_EXPORTED libusb_unlock_event_waiters (libusb_context * *ctx*)

Release the event waiters lock.

Parameters

ctx	the context to operate on, or NULL for the default context Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_wait_for_event (libusb_context * *ctx*, struct timeval * *tv*)

Wait for another thread to signal completion of an event.

Must be called with the event waiters lock held, see libusb_lock_event_waiters().

This function will block until any of the following conditions are met:

1. The timeout expires
2. A transfer completes
3. A thread releases the event handling lock through libusb_unlock_events()

Condition 1 is obvious. Condition 2 unblocks your thread *after* the callback for the transfer has completed. Condition 3 is important because it means that the thread that was previously handling events is no longer doing so, so if any events are to complete, another thread needs to step up and start event handling.

This function releases the event waiters lock before putting your thread to sleep, and reacquires the lock as it is being woken up.

Parameters

ctx	the context to operate on, or NULL for the default context
tv	maximum timeout for this blocking function. A NULL value indicates unlimited timeout.

Returns

0 after a transfer completes or another thread stops event handling
1 if the timeout expired Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_handle_events_timeout_completed (libusb_context * *ctx*, struct timeval * *tv*, int * *completed*)

Handle any pending events.

libusb determines "pending events" by checking if any timeouts have expired and by checking the set of file descriptors for activity.

If a zero timeval is passed, this function will handle any already-pending events and then immediately return in non-blocking style.

If a non-zero timeval is passed and no events are currently pending, this function will block waiting for events to handle up until the specified timeout. If an event arrives or a signal is raised, this function will return early.

If the parameter completed is not NULL then *after obtaining the event handling lock* this function will return immediately if the integer pointed to is not 0. This allows for race free waiting for the completion of a specific transfer.

Parameters

ctx	the context to operate on, or NULL for the default context
tv	the maximum time to block waiting for events, or an all zero timeval struct for non-blocking mode
completed	pointer to completion integer to check, or NULL

Returns

0 on success, or a LIBUSB_ERROR code on failure Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_handle_events_timeout (libusb_context * *ctx*, struct timeval * *tv*)

Handle any pending events.

Like libusb_handle_events_timeout_completed(), but without the completed parameter, calling this function is equivalent to calling libusb_handle_events_timeout_completed() with a NULL completed parameter.

This function is kept primarily for backwards compatibility. All new code should call libusb_handle_events_completed() or libusb_handle_events_timeout_completed() to avoid race conditions.

Parameters

ctx	the context to operate on, or NULL for the default context
tv	the maximum time to block waiting for events, or an all zero timeval struct for non-blocking mode

Returns

0 on success, or a LIBUSB_ERROR code on failure

int API_EXPORTED libusb_handle_events (libusb_context * *ctx*)

Handle any pending events in blocking mode.

There is currently a timeout hardcoded at 60 seconds but we plan to make it unlimited in future. For finer control over whether this function is blocking or non-blocking, or for control over the timeout, use libusb_handle_events_timeout_completed() instead.

This function is kept primarily for backwards compatibility. All new code should call libusb_handle_events_completed() or libusb_handle_events_timeout_completed() to avoid race conditions.

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

0 on success, or a LIBUSB_ERROR code on failure

int API_EXPORTED libusb_handle_events_completed (libusb_context * *ctx*, int * *completed*)

Handle any pending events in blocking mode.

Like libusb_handle_events(), with the addition of a completed parameter to allow for race free waiting for the completion of a specific transfer.

See libusb_handle_events_timeout_completed() for details on the completed parameter.

Parameters

ctx	the context to operate on, or NULL for the default context
completed	pointer to completion integer to check, or NULL

Returns

0 on success, or a LIBUSB_ERROR code on failure Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_handle_events_locked (libusb_context * *ctx*, struct timeval * *tv*)

Handle any pending events by polling file descriptors, without checking if any other threads are already doing so.

Must be called with the event lock held, see libusb_lock_events().

This function is designed to be called under the situation where you have taken the event lock and are calling poll()/select() directly on libusb's file descriptors (as opposed to using libusb_handle_events() or similar). You detect events on libusb's descriptors, so you then call this function with a zero timeout value (while still holding the event lock).

Parameters

ctx	the context to operate on, or NULL for the default context
tv	the maximum time to block waiting for events, or zero for non-blocking mode

Returns

0 on success, or a LIBUSB_ERROR code on failure Multi-threaded applications and asynchronous I/O

int API_EXPORTED libusb_pollfds_handle_timeouts (libusb_context * *ctx*)

Determines whether your application must apply special timing considerations when monitoring libusb's file descriptors.

This function is only useful for applications which retrieve and poll libusb's file descriptors in their own main loop (The more advanced option).

Ordinarily, libusb's event handler needs to be called into at specific moments in time (in addition to times when there is activity on the file descriptor set). The usual approach is to use libusb_get_next_timeout() to learn about when the next timeout occurs, and to adjust your poll()/select() timeout accordingly so that you can make a call into the library at that time.

Some platforms supported by libusb do not come with this baggage - any events relevant to timing will be represented by activity on the file descriptor set, and libusb_get_next_timeout() will always return 0. This function allows you to detect whether you are running on such a platform.

Since v1.0.5.

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

0 if you must call into libusb at times determined by libusb_get_next_timeout(), or 1 if all timeout events are handled internally or through regular activity on the file descriptors. Polling libusb file descriptors for event handling

int API_EXPORTED libusb_get_next_timeout (libusb_context * *ctx*, struct timeval * *tv*)

Determine the next internal timeout that libusb needs to handle.

You only need to use this function if you are calling poll() or select() or similar on libusb's file descriptors yourself - you do not need to use it if you are calling libusb_handle_events() or a variant directly.

You should call this function in your main loop in order to determine how long to wait for select() or poll() to return results. libusb needs to be called into at this timeout, so you should use it as an upper bound on your select() or poll() call.

When the timeout has expired, call into libusb_handle_events_timeout() (perhaps in non-blocking mode) so that libusb can handle the timeout.

This function may return 1 (success) and an all-zero timeval. If this is the case, it indicates that libusb has a timeout that has already expired so you should call libusb_handle_events_timeout() or similar immediately. A return code of 0 indicates that there are no pending timeouts.

On some platforms, this function will always returns 0 (no pending timeouts). See Notes on time-based events.

Parameters

ctx	the context to operate on, or NULL for the default context
tv	output location for a relative time against the current clock in which libusb must be called into in order to process timeout events

Returns

0 if there are no pending timeouts, 1 if a timeout was returned, or LIBUSB_ERROR_OTHER on failure

void API_EXPORTED libusb_set_pollfd_notifiers (libusb_context * *ctx*, libusb_pollfd_added_cb *added_cb*, libusb_pollfd_removed_cb *removed_cb*, void * *user_data*)

Register notification functions for file descriptor additions/removals.

These functions will be invoked for every new or removed file descriptor that libusb uses as an event source.

To remove notifiers, pass NULL values for the function pointers.

Note that file descriptors may have been added even before you register these notifiers (e.g. at libusb_init() time).

Additionally, note that the removal notifier may be called during libusb_exit() (e.g. when it is closing file descriptors that were opened and added to the poll set at libusb_init() time). If you don't want this, remove the notifiers immediately before calling libusb_exit().

Parameters

ctx	the context to operate on, or NULL for the default context

added_cb	pointer to function for addition notifications
removed_cb	pointer to function for removal notifications
user_data	User data to be passed back to callbacks (useful for passing context information)

DEFAULT_VISIBILITY const struct libusb_pollfd∗∗ LIBUSB_CALL libusb_get_pollfds (libusb_context ∗ *ctx*)

Retrieve a list of file descriptors that should be polled by your main loop as libusb event sources.

The returned list is NULL-terminated and should be freed with libusb_free_pollfds() when done. The actual list contents must not be touched.

As file descriptors are a Unix-specific concept, this function is not available on Windows and will always return NULL.

Parameters

ctx	the context to operate on, or NULL for the default context

Returns

a NULL-terminated list of libusb_pollfd structures
NULL on error
NULL on platforms where the functionality is not available

void API_EXPORTED libusb_free_pollfds (const struct libusb_pollfd ∗∗ *pollfds*)

Free a list of libusb_pollfd structures.

This should be called for all pollfd lists allocated with libusb_get_pollfds().

Since version 1.0.20, LIBUSB_API_VERSION >= 0x01000104

It is legal to call this function with a NULL pollfd list. In this case, the function will simply return safely.

Parameters

pollfds	the list of libusb_pollfd structures to free

31.8 Synchronous device I/O

This page documents libusb's synchronous (blocking) API for USB device I/O.

Functions

- int API_EXPORTED libusb_control_transfer (libusb_device_handle *dev_handle, uint8_t bmRequestType, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char *data, uint16_t wLength, unsigned int timeout)

  Perform a USB control transfer.
- int API_EXPORTED libusb_bulk_transfer (struct libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *transferred, unsigned int timeout)

  Perform a USB bulk transfer.
- int API_EXPORTED libusb_interrupt_transfer (struct libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *transferred, unsigned int timeout)

  Perform a USB interrupt transfer.

31.8.1 Detailed Description

This page documents libusb's synchronous (blocking) API for USB device I/O. This interface is easy to use but has some limitations. More advanced users may wish to consider using the asynchronous I/O API instead.

31.8.2 Function Documentation

int API_EXPORTED libusb_control_transfer (libusb_device_handle * *dev_handle,* uint8_t *bmRequestType,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char * *data,* uint16_t *wLength,* unsigned int *timeout*)

Perform a USB control transfer.

The direction of the transfer is inferred from the bmRequestType field of the setup packet.

The wValue, wIndex and wLength fields values should be given in host-endian byte order.

Parameters

dev_handle	a handle for the device to communicate with
bmRequestType	the request type field for the setup packet
bRequest	the request field for the setup packet
wValue	the value field for the setup packet
wIndex	the index field for the setup packet
data	a suitably-sized data buffer for either input or output (depending on direction bits within bmRequestType)
wLength	the length field for the setup packet. The data buffer should be at least this size.
timeout	timeout (in millseconds) that this function should wait before giving up due to no response being received. For an unlimited timeout, use value 0.

Returns

on success, the number of bytes actually transferred
LIBUSB_ERROR_TIMEOUT if the transfer timed out
LIBUSB_ERROR_PIPE if the control request was not supported by the device
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_BUSY if called from event handling context
LIBUSB_ERROR_INVALID_PARAM if the transfer size is larger than the operating system and/or hardware can support
another LIBUSB_ERROR code on other failures

int API_EXPORTED libusb_bulk_transfer (struct libusb_device_handle * *dev_handle,* unsigned char *endpoint,* unsigned char * *data,* int *length,* int * *transferred,* unsigned int *timeout*)

Perform a USB bulk transfer.

The direction of the transfer is inferred from the direction bits of the endpoint address.

For bulk reads, the `length` field indicates the maximum length of data you are expecting to receive. If less data arrives than expected, this function will return that data, so be sure to check the `transferred` output parameter.

You should also check the `transferred` parameter for bulk writes. Not all of the data may have been written.

Also check `transferred` when dealing with a timeout error code. libusb may have to split your transfer into a number of chunks to satisfy underlying O/S requirements, meaning that the timeout may expire after the first few chunks have completed. libusb is careful not to lose any data that may have been transferred; do not assume that timeout conditions indicate a complete lack of I/O.

Parameters

dev_handle	a handle for the device to communicate with
endpoint	the address of a valid endpoint to communicate with
data	a suitably-sized data buffer for either input or output (depending on endpoint)
length	for bulk writes, the number of bytes from data to be sent. for bulk reads, the maximum number of bytes to receive into the data buffer.
transferred	output location for the number of bytes actually transferred. Since version 1.0.21 (LIBUSB_API_VERSION >= 0x01000105), it is legal to pass a NULL pointer if you do not wish to receive this information.
timeout	timeout (in millseconds) that this function should wait before giving up due to no response being received. For an unlimited timeout, use value 0.

Returns

0 on success (and populates `transferred`)
LIBUSB_ERROR_TIMEOUT if the transfer timed out (and populates `transferred`)
LIBUSB_ERROR_PIPE if the endpoint halted
LIBUSB_ERROR_OVERFLOW if the device offered more data, see Packets and overflows
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_BUSY if called from event handling context
another LIBUSB_ERROR code on other failures

int API_EXPORTED libusb_interrupt_transfer (struct libusb_device_handle * *dev_handle,* unsigned char *endpoint,* unsigned char * *data,* int *length,* int * *transferred,* unsigned int *timeout*)

Perform a USB interrupt transfer.

The direction of the transfer is inferred from the direction bits of the endpoint address.

For interrupt reads, the `length` field indicates the maximum length of data you are expecting to receive. If less data arrives than expected, this function will return that data, so be sure to check the `transferred` output parameter.

You should also check the `transferred` parameter for interrupt writes. Not all of the data may have been written.

Also check `transferred` when dealing with a timeout error code. libusb may have to split your transfer into a number of chunks to satisfy underlying O/S requirements, meaning that the timeout may expire after the first few chunks have completed. libusb is careful not to lose any data that may have been transferred; do not assume that timeout conditions indicate a complete lack of I/O.

The default endpoint bInterval value is used as the polling interval.

Parameters

dev_handle	a handle for the device to communicate with
endpoint	the address of a valid endpoint to communicate with
data	a suitably-sized data buffer for either input or output (depending on endpoint)
length	for bulk writes, the number of bytes from data to be sent. for bulk reads, the maximum number of bytes to receive into the data buffer.
transferred	output location for the number of bytes actually transferred. Since version 1.0.21 (LIBUSB_API_VERSION >= 0x01000105), it is legal to pass a NULL pointer if you do not wish to receive this information.
timeout	timeout (in millseconds) that this function should wait before giving up due to no response being received. For an unlimited timeout, use value 0.

Returns

0 on success (and populates `transferred`)
LIBUSB_ERROR_TIMEOUT if the transfer timed out
LIBUSB_ERROR_PIPE if the endpoint halted
LIBUSB_ERROR_OVERFLOW if the device offered more data, see Packets and overflows
LIBUSB_ERROR_NO_DEVICE if the device has been disconnected
LIBUSB_ERROR_BUSY if called from event handling context
another LIBUSB_ERROR code on other error

Chapter 32

Namespace Documentation

32.1 AIOUSB Namespace Reference

Data Structures

- class AI16_DataPoint

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

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

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

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

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

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

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

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

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

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

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

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

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

  Class USB_AIO16_Family represents a USB-AI16-family device, which encompasses the following product IDs: USB_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, 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)

32.1.1 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 33

Data Structure Documentation

33.1 ad_gain_pairs Struct Reference

```
#include <libaiousb.h>
```

Data Fields

- ADGainCode gain
- const char * name

33.1.1 Field Documentation

ADGainCode gain

const char * name

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

- lib/AIOChannelRange.h
- lib/ndkbuild/jni/libaiousb.h

33.2 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.

33.2.1 Field Documentation

AIOUSBDevice * device

Pointer to the device Descriptor.

unsigned long size

unsigned char registers

unsigned timeout

ADCMuxSettings mux_settings

int clock_rate

AIOUSB_BOOL discardFirstSample

AIOUSB_BOOL debug

AIOUSB_BOOL testing

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

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

- lib/ADCConfigBlock.h

33.3 ADRange Struct Reference

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

Data Fields

- double minVolts
- double range

33.3.1 Field Documentation

double minVolts

double range

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

- lib/AIOUSB_Core.h

33.4 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 >

33.4.1 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.

33.4.2 Constructor & Destructor Documentation

AI16_DataPoint ( )

33.4.3 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.

33.4.4 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]`

33.4.5 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

33.5 AI16_DataPointArray Class Reference

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

Public Member Functions

- AI16_DataPointArray (int size=0)

33.5.1 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

33.6 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

33.6.1 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)

33.6.2 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.

33.6.3 Member Function Documentation

AnalogInputSubsystem& getSubsystem ( ) [inline]

Gets the subsystem from which this data set was obtained.

Returns

The subsystem from which this data set was obtained.

const AI16_DataPointArray& getPoints ( ) [inline]

Gets the data point array from this data set.

Returns

The data point array from this data set.

long getTimeStamp ( ) [inline]

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

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

Returns

The approximate time stamp when this data set was captured.

int getCalMode ( ) [inline]

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

Returns

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

See Also

AnalogInputSubsystem::getCalMode() const

int getTriggerMode ( ) [inline]

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

Returns

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

See Also

AnalogInputSubsystem::getTriggerMode() const

int getOverSample ( ) [inline]

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

Returns

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

See Also

AnalogInputSubsystem::getOverSample() const

bool isDiscardFirstSample ( ) [inline]

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

Returns

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

See Also

AnalogInputSubsystem::isDiscardFirstSample() const

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

Prints this data set.

Mainly useful for diagnostic purposes.

Parameters

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

Returns

The print stream.

33.6.4 Friends And Related Function Documentation

friend class AnalogInputSubsystem [friend]

33.6.5 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

33.7 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

33.7.1 Constructor & Destructor Documentation

AI16_InputRange ( ) [protected]

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

~AI16_InputRange ( ) [protected], [virtual]

33.7.2 Member Function Documentation

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

Sets the range ID.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

33.7.3 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

33.8 aio_callback_args Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- struct libusb_transfer * transfer
- struct aiocb aiocb

33.8.1 Field Documentation

struct libusb_transfer* transfer

struct aiocb aiocb

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.9 aio_channel_range Struct Reference

```
#include <libaiousb.h>
```

Data Fields

- int start
- int end
- ADGainCode gain

33.9.1 Field Documentation

int start

int end

ADGainCode gain

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

- lib/AIOChannelRange.h
- lib/ndkbuild/jni/libaiousb.h

33.10 aio_counts_converter Struct Reference

```
#include <libaiousb.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

33.10.1 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 files:

- lib/AIOCountsConverter.h
- lib/ndkbuild/jni/libaiousb.h

33.11 aio_either_val Struct Reference

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

Data Fields

- AIO_NUMBER number
- void ∗ object

33.11.1 Field Documentation

AIO_NUMBER number

void ∗ **object**

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

- lib/AIOEither.h

33.12 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

33.12.1 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

33.13 AIOArgument Struct Reference

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

Data Fields

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

33.13.1 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

33.14 AIOArguments Struct Reference

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

Data Fields

- AIOArgument * device_args
- int number_arguments

33.14.1 Field Documentation

AIOArgument * device_args

int number_arguments

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

- lib/AIOConfiguration.h

33.15 AIOBuf Struct Reference

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

Data Fields

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

33.15.1 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

33.16 aiobuf_iterator Struct Reference

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

Data Fields

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

33.16.1 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

33.17 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

33.17.1 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

33.18 AIOChannelRangeTmp Struct Reference

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

Data Fields

- int start_channel
- int end_channel
- int gaincode

33.18.1 Field Documentation

int start_channel

int end_channel

int gaincode

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

- lib/AIOCommandLine.h

33.19 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

33.19.1 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

33.20 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

33.20.1 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

33.21 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)

33.21.1 Detailed Description

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

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

- Create a new AIOContinuousBuf of a certain size that is large enough to handle the running clock rate * number_of_oversamples *
- Assign a device index to the AIOContinuousBuf
- Start am acquisition by calling AIOContinuousBufInitiateCallbackAcquisition;
- Process the input data using either a simple while loop burst_test.c

  or using the callback function as in

33.21.2 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

33.22 AIODeviceInfo Struct Reference

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

Data Fields

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

33.22.1 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

33.23 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.

33.23.1 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

33.24 aioerror Struct Reference

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

Data Fields

- AIORET_TYPE retval
- char * error_message

33.24.1 Field Documentation

AIORET_TYPE retval

char * error_message

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

- lib/AIOUSB_Core.h

33.25 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.

33.25.1 Detailed Description

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

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

33.25.2 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

33.26 AIOGainRange Struct Reference

```
#include <libaiousb.h>
```

Data Fields

- double min
- double max

33.26.1 Field Documentation

double min

double max

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

- lib/AIOCountsConverter.h
- lib/ndkbuild/jni/libaiousb.h

33.27 AIOProductGroup Struct Reference

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

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

33.27.1 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

33.28 AIOProductRange Struct Reference

A simplified range of Products based off of device ids.

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

33.28.1 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

33.29 aiousb_libusb_args Struct Reference

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

Data Fields

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

33.29.1 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/ndkbuild/jni/USBDevice.h

33.30 AIOUSBDevice Struct Reference

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

Data Fields

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

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

  timeout for device communication (ms.)
- double miscClockHz

  miscellaneous clock frequency setting
- unsigned ProductID

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

  number of A/D channels in each config.
- AIOUSB_BOOL bDIOStream
- unsigned long StreamingBlockSize
- AIOUSB_BOOL bDIODebounce
- AIOUSB_BOOL bDIOSPI
- AIOUSB_BOOL bSetCustomClocks
- unsigned WDGBytes
- AIOUSB_BOOL bClearFIFO
- unsigned ImmADCs
- AIOUSB_BOOL bDACBoardRange
- AIOUSB_BOOL bDACChannelCal
- unsigned FlashSectors
- AIOUSB_BOOL bDACOpen
- AIOUSB_BOOL bDACClosing
- AIOUSB_BOOL bDACAborting
- AIOUSB_BOOL bDACStarted
- unsigned char ∗∗ DACData
- unsigned char ∗ PendingDACData
- pthread_mutex_t hDACDataMutex
- sem_t hDACDataSem
- AIOUSB_BOOL bDIOOpen
- AIOUSB_BOOL bDIORead
- AIOUSB_BOOL bDeviceWasHere
- unsigned char ∗ LastDIOData
- char ∗ cachedName
- unsigned long cachedSerialNumber
- ADCConfigBlock cachedConfigBlock

  .size == 0 == uninitialized
- AIOUSB_BOOL workerBusy

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

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

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

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

33.30.1 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

33.31 aiousboption Struct Reference

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

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

- lib/AIOUSB_Core.h

33.32 AIOWDGConfig Struct Reference

```
#include <libaiousb.h>
```

Data Fields

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

33.32.1 Field Documentation

int bufsize

unsigned long L

unsigned char ∗ wdgbuf

unsigned long timeout

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

- lib/AIOUSB_WDG.h
- lib/ndkbuild/jni/libaiousb.h

33.33 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
- AI16_InputRange ∗ inputRange
- bool ∗ differentialMode
- int calMode
- int triggerMode
- int startChannel
- int endChannel
- int overSample
- unsigned short ∗ readBulkBuffer
- int readBulkSamplesRequested
- int readBulkSamplesRetrieved
- bool autoConfig

Static Protected Attributes

- static const char RANGE_TEXT [][10]
- static const int NUM_CONFIG_REGISTERS = 20
- static const int NUM_MUX_CONFIG_REGISTERS = 21
- static const int NUM_GAIN_CODE_REGISTERS = 16
- static const int REG_GAIN_CODE = 0
- static const int REG_CAL_MODE = 16
- static const int REG_TRIG_MODE = 17
- static const int REG_START_END = 18
- static const int REG_OVERSAMPLE = 19
- static const int REG_MUX_START_END = 20

- static const int DIFFERENTIAL_MODE = 8
- static const int MAX_OVERSAMPLE = 0xff
- static const int TRIG_MODE_VALID_MASK
- static const int AUTO_CAL_UNKNOWN = 0
- static const int AUTO_CAL_NOT_PRESENT = 1
- static const int AUTO_CAL_PRESENT = 2
- static const int MAX_CHANNELS = 128

Friends

- class USB_AI16_Family

Additional Inherited Members

33.33.1 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())*.

33.33.2 Constructor & Destructor Documentation

AnalogInputSubsystem (USBDeviceBase & *parent*)

~AnalogInputSubsystem ( ) `[virtual]`

33.33.3 Member Function Documentation

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

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

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumChannels ( ) const `[inline]`

Gets the number of primary analog input channels.

Returns

Number of channels, numbered 0 to n-1.

int getNumMUXChannels ( ) const `[inline]`

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

Returns

Number of channels, numbered 0 to n-1.

int getChannelsPerGroup ( ) `[inline]`

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

Returns

The number of channels per group.

bool isAutoCalPresent (bool *force*)

Tells if automatic calibration is possible with this device.

Parameters

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

Returns

True indicates that automatic calibration is available.

See Also

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

Exceptions

OperationFailedException	

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

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

bool isAutoConfig ( ) const `[inline]`

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

Returns

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

See Also

setAutoConfig(bool autoConfig)

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

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

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

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

AnalogInputSubsystem & readConfig ( )

Reads the A/D configuration from the device.

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

AnalogInputSubsystem & writeConfig ( )

Writes the A/D configuration to the device.

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

bool isDiscardFirstSample ( ) const

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

Returns

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

AnalogInputSubsystem & setDiscardFirstSample (bool *discard*)

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

int getCalMode ( ) const `[inline]`

Gets the current calibration mode.

Returns

Current calibration mode (*AnalogInputSubsystem::CAL_MODE_NORMAL*, *AnalogInputSubsystem::CAL_MODE_GROUND* or *AnalogInputSubsystem::CAL_MODE_REFERENCE*).

See Also

setCalMode(int calMode)

AnalogInputSubsystem & setCalMode (int *calMode*)

Sets the A/D calibration mode.

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

Parameters

calMode	the calibration mode. May be one of: *AnalogInputSubsystem::CAL_MODE_NORMAL* *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

triggerMode	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

channel	the channel for which to obtain the current range.

Returns

Current range for *channel*.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

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

Gets the current range for multiple A/D channels.

Parameters

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

Returns

Array containing the current ranges for the specified channels.

See Also

setRange(int startChannel, const IntArray &range)

Exceptions

IllegalArgumentException	

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

Sets the range for a single A/D channel.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setDifferentialMode(int channel, bool differentialMode)

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

Sets the range for multiple A/D channels.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

bool isDifferentialMode (int *channel*) const

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

Parameters

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

Returns

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

See Also

setDifferentialMode(int channel, boolean differentialMode)

Exceptions

IllegalArgumentException	

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

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

Parameters

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

Returns

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

See Also

setDifferentialMode(int startChannel, const BoolArray &differentialMode)

Exceptions

IllegalArgumentException	

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

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

When using differential mode, one should have a good understanding of how the hardware implements it. Considering the simple case of a device with only sixteen input channels, when differential mode is enabled for a channel, that channel is paired with another channel, eight higher than the one for which differential mode is enabled. For instance, if differential mode is enabled for channel 1, then it is paired with channel 9, meaning that channel 1 will return the voltage difference between channels 1 and 9, and channel 9 will no longer return a meaningful reading. This scheme also means that enabling differential mode for channels 8-15 has no effect. In fact, if one attempts to enable differential mode for channels 8-15, nothing happens and if the differential mode setting is read back from the device for those channels, it will likely no longer be enabled! Further confusing matters is that some newer firmware does not clear the differential mode setting for channels 8-15, meaning that it will be returned from the device exactly as set even though it has no effect. So ... one should not rely on the differential mode setting for channels 8-15 to behave in a consistent

or predictable manner. For consistency and simplicity, one may read counts or volts from channels 8-15 even while differential mode is enabled, but the readings will not be meaningful. In differential mode, only the base channel (0-7) of the pair that's enabled for differential mode will return a meaningful reading. Channels 8-15 which are not enabled for differential mode will continue to return meaningful readings. For example, if differential mode is enabled for channel 1, then channel 1 will return a meaningful reading, channel 9 will not, and channels 8 and 10-15 will. Considering the more complex case of a device such as the USB-AI16-64MA, which has an additional MUX affording 32 differential, or 64 single-ended inputs, things are a bit more complex. In this case, channels 0-3 share the same differential mode (and range) setting; channels 4-7 share the same setting; and so on. For the sake of simplicity and to support future designs which may have distinct settings for all channels, this software permits the differential mode (and range) to be specified for *any* MUXed channel, even though ultimately multiple channels may share the same setting. For example, on such a device as this, setting the differential mode (or range) of channel 1 also sets the differential mode (or range) of channels 0, 2 and 3. There is yet another case to consider, that of devices such as the USB-AI16-128A. This device may have up to 128 channels, which share settings in groups of eight rather than four on the USB-AI16-64MA. Method *getChannelsPerGroup()* tells how many channels are grouped together on each device, and this topic is discussed more thoroughly in `http://accesio.com/MANUALS/USB-AI FAMILY.PDF`. The foregoing description also applies to the range setting, so one should refer to *setRange(int channel, int range)* as well.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

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

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

Parameters

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

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

int getOverSample ( ) const `[inline]`

Gets the current number of over-samples.

Returns

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

See Also

setOverSample(int oversample)

AnalogInputSubsystem & setOverSample (int *overSample*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

AnalogInputSubsystem & setCalibrationTable (const UShortArray & *calTable*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

int getStreamingBlockSize ( ) [inline]

Gets the current streaming block size.

Returns

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

Exceptions

OperationFailedException	

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

Sets the streaming block size.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double getClock ( ) [inline]

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

Returns

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

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

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

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

Parameters

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

Returns

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

Exceptions

OperationFailedException	

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

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

The proper way to use this function is to configure the A/D with a default calibration table (such as by calling *calibrate(bool autoCal, bool returnCalTable, const std::string &saveFileName)*). Then inject a series of voltages into one of the A/D input channels, recording the count values reported by the A/D (by calling *readCounts(int channel)*). It's also a good idea to enable oversampling while recording these values in order to obtain the most stable readings. Alternatively, since *points* is an array of *double* values, you can obtain individual A/D count measurements and average them yourself, producing a *double* average, and put that value into the *points* array. The *points* array consists of voltage-count pairs; *points[0]* is the first input voltage; *points[1]* is the corresponding count value measured by the A/D; *points[2]* and *points[3]* contain the second pair of voltage-count values; and so on. You can provide any number of pairs, although more than a few dozen is probably overkill, not to mention would take a lot of effort to acquire. This calibration procedure uses the current gain A/D setting for channel 0, so it must be the same as that used to collect the measured A/D counts. It's recommended that all the channels be set to the same gain, the one that will be used during normal operation. The calibration is gain dependent, so switching the gain after calibrating may introduce slight offset or gain changes. So for best results, the A/D should be calibrated on the same gain setting that will be used during normal operation. You can create any number of calibration tables. If your application needs to switch between ranges, you may wish to create a separate calibration table for each range your application will use. Then when switching to a different range, the application can load the appropriate calibration table. Although calibrating in this manner does take some effort, it produces the best results, eliminating all sources of error from the input pins onward. Furthermore, the calibration table can be saved to a file and reloaded into the A/D, ensuring consistency.

Parameters

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

Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

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

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

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

Parameters

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

Returns

A data set containing the samples and the sampling parameters.

Exceptions

OperationFailedException	

unsigned short readCounts (int *channel*)

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

Parameters

channel	the channel to read.

Returns

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

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

Reads the A/D count values from multiple channels.

Parameters

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

Returns

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

Exceptions

OperationFailedException	

double readVolts (int *channel*)

Reads the voltage from a single channel.

Parameters

channel	the channel to read.

Returns

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

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

Reads the voltage from multiple channels.

Parameters

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

Returns

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

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

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

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

int readBulkSamplesAvailable ( )

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

Returns

The number of samples available.

Exceptions

OperationFailedException	

UShortArray readBulkNext (int *numSamples*)

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

Parameters

numSamples	the number of samples to retrieve.

Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

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

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

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

Parameters

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

Returns

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

Exceptions

IllegalArgumentException	

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

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

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

Parameters

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

Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

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

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

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

Parameters

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

Returns

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

Exceptions

IllegalArgumentException	

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

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

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

Parameters

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

Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

33.33.4 Friends And Related Function Documentation

friend class USB_AI16_Family `[friend]`

33.33.5 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

33.34 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

33.34.1 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.

33.34.2 Constructor & Destructor Documentation

AnalogIORange ( )

Constructor which uses the default properties.

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

Constructor which sets the count range.

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

Parameters

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

~**AnalogIORange** ( ) [virtual]

33.34.3 Member Function Documentation

int getRange ( ) **const** [inline]

Gets the current range ID.

Returns

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

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

Sets the range ID.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented in AI16_InputRange, AO16_OutputRange, and DA12_OutputRange.

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

Sets the voltage range.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

double countsToVolts (int *counts*) const

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

Parameters

counts	the count value to convert to volts.

Returns

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

int voltsToCounts (double *volts*) const

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

Parameters

volts	the voltage value to convert to counts.

Returns

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

33.34.4 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

33.35 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

33.35.1 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())*.

33.35.2 Constructor & Destructor Documentation

AnalogOutputSubsystem (USBDeviceBase & *parent*)

~AnalogOutputSubsystem ( ) `[virtual]`

33.35.3 Member Function Documentation

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

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumChannels ( ) const `[inline]`

Gets the number of analog output channels.

Returns

Number of channels, numbered 0 to n-1.

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

Writes a count value to a D/A channel.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

AnalogOutputSubsystem & writeCounts (const UShortArray & *points*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

33.35.4 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

33.36 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

33.36.1 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())*.

33.36.2 Constructor & Destructor Documentation

AO16_AnalogOutputSubsystem (USBDeviceBase & *parent*) `[protected]`

~AO16_AnalogOutputSubsystem ( ) `[protected],[virtual]`

33.36.3 Member Function Documentation

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

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int range)

Exceptions

IllegalArgumentException	

int getRange ( ) const `[inline]`

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

Returns

Current voltage range.

See Also

setRange(int range)

AO16_AnalogOutputSubsystem & setRange (int *range*)

Sets the voltage range of the D/A outputs.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

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

Writes a voltage value to a D/A channel.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

AO16_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray & *points*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

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

Parameters

counts	the count value to convert to volts.

Returns

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

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

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

Parameters

volts	the voltage value to convert to counts.

Returns

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

33.36.4 Friends And Related Function Documentation

friend class USB_AO16_Family `[friend]`

33.36.5 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

33.37 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

33.37.1 Constructor & Destructor Documentation

AO16_OutputRange ( ) [protected]

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

~AO16_OutputRange ( ) [protected],[virtual]

33.37.2 Member Function Documentation

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

Sets the range ID.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

33.37.3 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

33.38 BLooper Class Reference

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

- lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp

33.39 BoolArray Class Reference

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

Public Member Functions

- BoolArray (int size=0)

33.39.1 Constructor & Destructor Documentation

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

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

- deprecated/classlib/USBDeviceBase.hpp

33.40 BulkAcquireWorkerParams Struct Reference

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

Data Fields

- unsigned long DeviceIndex
- unsigned long BufSize
- void * pBuf

33.40.1 Field Documentation

unsigned long DeviceIndex

unsigned long BufSize

void * pBuf

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

- lib/AIOUSB_Core.h

33.41 channel_range Struct Reference

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

Data Fields

- int start_channel
- int end_channel
- int gaincode

33.41.1 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

33.42 cJSON Struct Reference

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

Data Fields

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

33.42.1 Field Documentation

struct cJSON * next

struct cJSON * prev

struct cJSON * child

int type

char * valuestring

int valueint

double valuedouble

char * string

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

- lib/cJSON.h

33.43 cJSON_Hooks Struct Reference

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

Data Fields

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

33.43.1 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

33.44 command_block_wrapper Struct Reference

Data Fields

- uint8_t dCBWSignature [4]
- uint32_t dCBWTag
- uint32_t dCBWDataTransferLength

- uint8_t bmCBWFlags
- uint8_t bCBWLUN
- uint8_t bCBWCBLength
- uint8_t CBWCB [16]

33.44.1 Field Documentation

uint8_t dCBWSignature[4]

uint32_t dCBWTag

uint32_t dCBWDataTransferLength

uint8_t bmCBWFlags

uint8_t bCBWLUN

uint8_t bCBWCBLength

uint8_t CBWCB[16]

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

- lib/ndkbuild/libusb/examples/xusb.c

33.45 command_status_wrapper Struct Reference

Data Fields

- uint8_t dCSWSignature [4]
- uint32_t dCSWTag
- uint32_t dCSWDataResidue
- uint8_t bCSWStatus

33.45.1 Field Documentation

uint8_t dCSWSignature[4]

uint32_t dCSWTag

uint32_t dCSWDataResidue

uint8_t bCSWStatus

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

- lib/ndkbuild/libusb/examples/xusb.c

33.46 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

33.46.1 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

33.47 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 *)

33.47.1 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

33.48 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

33.48.1 Detailed Description

Class Counter represents a single counter/timer.

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

33.48.2 Constructor & Destructor Documentation

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

33.48.3 Member Function Documentation

int getDeviceIndex ( ) const

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

Used internally in calls to the underlying API.

Returns

The index of the parent device on the USB bus.

Counter & setMode (int *mode*)

Sets the counter's mode.

Parameters

mode	the counter mode. May be one of: *Counter::MODE_TERMINAL_COUNT Counter::MODE_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

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

Returns

This counter, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

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

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

Parameters

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

Returns

This counter, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

unsigned short readCount ( )

Reads a counter's current count value.

Returns

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

Exceptions

OperationFailedException	

UShortArray readCountAndStatus ( )

Reads a counter's current count value and status.

Returns

An array of 2 16-bit integers: char[0] contains the current count value (0-65,535) char[1] contains the current counter status (0-255)

Exceptions

OperationFailedException	

unsigned short readCountAndSetModeAndCount (int *mode*, unsigned short *count*)

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

Parameters

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

Returns

The 16-bit count value (0-65,535) prior to setting the new mode and count.

Exceptions

IllegalArgumentException	
OperationFailedException	

33.48.4 Friends And Related Function Documentation

friend class CounterSubsystem `[friend]`

33.48.5 Field Documentation

const int MODE_TERMINAL_COUNT = 0 `[static]`

Mode 0: interrupt on terminal count *(see setMode(int mode))*.

const int MODE_ONE_SHOT = 1 `[static]`

Mode 1: hardware retriggerable one-shot *(see setMode(int mode))*.

const int MODE_RATE_GENERATOR = 2 `[static]`

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

const int MODE_SQUARE_WAVE = 3 `[static]`

Mode 3: square wave mode *(see setMode(int mode))*.

const int MODE_SW_TRIGGERED = 4 `[static]`

Mode 4: software triggered mode *(see setMode(int mode))*.

const int MODE_HW_TRIGGERED = 5 [static]

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

CounterSubsystem* parent [protected]

int counterIndex [protected]

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

- deprecated/classlib/Counter.hpp
- deprecated/classlib/Counter.cpp

33.49 CounterList Class Reference

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

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

- deprecated/classlib/Counter.hpp

33.50 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

33.50.1 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())*.

33.50.2 Constructor & Destructor Documentation

CounterSubsystem (USBDeviceBase & *parent*)

~CounterSubsystem ( ) `[virtual]`

33.50.3 Member Function Documentation

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

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumCounterBlocks ( ) const `[inline]`

Gets the number of counter blocks.

Typically there are three counters per counter block.

Returns

The number of counter blocks.

int getNumCounters ( ) const `[inline]`

Gets the number of individual counters, indexed from 0 to n-1.

Returns

The number of individual counters.

Counter & getCounter (int *counter*)

Gets a reference to an individual counter.

You must obtain a reference to a counter before you can perform counter operations.

Parameters

counter	the counter for which to obtain a reference (0 to n-1).

Returns

A reference to the specified counter.

Exceptions

IllegalArgumentException	

UShortArray readCounts (bool *oldData*)

Reads the current count values of all the counters, optionally including an "old data" indication.

If *oldData* is *true*, then an extra word will be returned (one word for each counter, plus one extra word) that contains an "old data" indication, which is useful for optimizing polling rates. If the value of the final word is zero, then the data is "old data," meaning you are polling the counters faster than your gate signal is running. If *oldData* is *false*, then only the count values are returned.

Parameters

oldData	*true* includes the "old data" indication in the returned data; *false* returns just the count values.

Returns

An array containing the current count values for all the counters plus an optional "old data" indication in the final word.

Exceptions

OperationFailedException	

CounterSubsystem & selectGate (int *counter*)

Selects the counter to use as a gate in frequency measurement on other counters.

Parameters

counter	the counter to select as a gate (0 to n-1).

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double startClock (int *counterBlock,* double *clockHz*)

Selects an output frequency for a counter block and starts the counters.

selectGate(int counter) and *readCounts(bool oldData)* are used in measuring frequency. To measure frequency one must count pulses for a known duration. In simplest terms, the number of pulses that occur for 1 second translates directly to Hertz. In the USB-CTR-15 and other supported devices, you can create a known duration by configuring one counter to act as a "gating" signal for any collection of other counters. The other "measurement" counters will only count during the "high" side of the gate signal, which we can control. So, to measure frequency you:

1. Create a gate signal of known duration
2. Connect this gating signal to the gate pins of all the "measurement" counters
3. Call *selectGate()* to tell the board which counter is generating that gate
4. Call *readCounts(true)* periodically to read the latched count values from all the "measurement" counters. In practice, it may not be possible to generate a gating signal of sufficient duration from a single counter. Simply concatenate two or more counters into a series, or daisy-chain, and use the last counter's output as your gating signal. This last counter in the chain should be selected as the "gate source" using *selectGate(int counter)*. Once a value has been read from a counter using the *readCounts(true)* call, it can be translated into actual Hz by dividing the count value returned by the high-side-duration of the gating signal, in seconds. For example, if your gate is configured for 10Hz, the high-side lasts 0.05 seconds. If you read 1324 counts via the *readCounts(true)* call, the frequency would be "1324 / 0.05", or 26.48 KHz.

   Parameters

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

   Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

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

Halts the counter started by *startClock(int counterBlock, double clockHz)*

Parameters

counterBlock	the counter block to halt generating a frequency.

Returns

This subsystem, useful for chaining together multiple operations.

33.50.4 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]`

33.50.5 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

33.51 CStringArray Struct Reference

`#include <CStringArray.h>`

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

- lib/CStringArray.h

33.52 DA12_AnalogOutputSubsystem Class Reference

Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device.

`#include <DA12_AnalogOutputSubsystem.hpp>`

Public Member Functions

- int getRange (int channel) const

  Gets the current voltage range of a D/A channel.
- IntArray getRange (int startChannel, int numChannels) const

  Gets the current voltage range of multiple D/A channels.
- DA12_AnalogOutputSubsystem & setRange (int channel, int range)

  Sets the voltage range of a D/A channel.
- DA12_AnalogOutputSubsystem & setRange (int startChannel, const IntArray &range)

  Sets the current voltage range of multiple D/A channels.

- DA12_AnalogOutputSubsystem & setRange (int range)

  Sets the current voltage range of all D/A channels to the same value.
- DA12_AnalogOutputSubsystem & writeVolts (int channel, double volts)

  Writes a voltage value to a D/A channel.
- DA12_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray &points)

  Writes a block of voltage values to one or more D/A channels.
- double countsToVolts (int channel, unsigned short counts) const

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

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

Static Public Member Functions

- static std::string getRangeText (int range)

  Gets the textual string for the specified range.

Static Public Attributes

- static const int RANGE_0_2_5V = 0

  Unipolar, 0-2.5 volt range (see setRange(int channel, int range)).
- static const int RANGE_2_5V = 1

  Bipolar, -2.5 to +2.5 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_5V = 2

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

  Bipolar, -5 to +5 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_10V = 4

  Unipolar, 0-10 volt range (see setRange(int channel, int range)).
- static const int RANGE_10V = 5

  Bipolar, -10 to +10 volt range (see setRange(int channel, int range)).
- static const int MIN_COUNTS = 0

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

  Maximum number of counts D/A can output.

Protected Member Functions

- DA12_AnalogOutputSubsystem (USBDeviceBase &parent)
- virtual ~DA12_AnalogOutputSubsystem ()

Protected Attributes

- DA12_OutputRange * outputRange

Static Protected Attributes

- static const char RANGE_TEXT [][10]

Friends

- class USB_DA12_8A_Family
- class USB_DA12_8E_Family

33.52.1 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())*.

33.52.2 Constructor & Destructor Documentation

DA12_AnalogOutputSubsystem (USBDeviceBase & *parent*) `[protected]`

~DA12_AnalogOutputSubsystem ( ) `[protected],[virtual]`

33.52.3 Member Function Documentation

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

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int range)

Exceptions

IllegalArgumentException	

int getRange (int *channel*) const

Gets the current voltage range of a D/A channel.

Parameters

channel	the channel for which to obtain the current range.

Returns

Current voltage range.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

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

Gets the current voltage range of multiple D/A channels.

Parameters

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

Returns

Array containing the current range for each of the specified channels.

See Also

setRange(int startChannel, const IntArray &range)

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *channel,* int *range*)

Sets the voltage range of a D/A channel.

The ranges in this device are selected by means of hardware jumpers, so these range settings here do not affect the hardware. However, they are used to perform conversions between volts and counts. Moreover, the range setting is per D/A channel, so care must be taken when setting the ranges to ensure that the software setting matches the hardware jumper configuration, otherwise the voltage-count conversions will be incorrect.

Parameters

channel	the channel for which to set the range.
range	the voltage range to select. May be one of: *DA12_AnalogOutputSubsystem::RANGE_0_2_5-V DA12_AnalogOutputSubsystem::RANGE_2_5V DA12_AnalogOutputSubsystem::RANGE_0-_5V DA12_AnalogOutputSubsystem::RANGE_5V DA12_AnalogOutputSubsystem::RANGE_0-_10V DA12_AnalogOutputSubsystem::RANGE_10V*

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *startChannel,* const IntArray & *range*)

Sets the current voltage range of multiple D/A channels.

Parameters

startChannel	the first channel for which to set the range.
range	an array of voltage ranges to select, one per channel. The length of this array implicitly specifies the number of channels to configure.

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *range*)

Sets the current voltage range of all D/A channels to the same value.

Parameters

range	the voltage range to select.

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setRange(int channel, int range)

DA12_AnalogOutputSubsystem & writeVolts (int *channel,* double *volts*)

Writes a voltage value to a D/A channel.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

DA12_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray & *points*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

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

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

Parameters

channel	the channel whose current range will be used to perform the conversion.
counts	the count value to convert to volts.

Returns

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

Exceptions

IllegalArgumentException	

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

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

Parameters

channel	the channel whose current range will be used to perform the conversion.
volts	the voltage value to convert to counts.

Returns

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

Exceptions

IllegalArgumentException	

33.52.4 Friends And Related Function Documentation

friend class USB_DA12_8A_Family `[friend]`

friend class USB_DA12_8E_Family `[friend]`

33.52.5 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

33.53 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

33.53.1 Constructor & Destructor Documentation

DA12_OutputRange ( ) [protected]

DA12_OutputRange (int *minCounts*, int *maxCounts*) [protected]

~**DA12_OutputRange** ( ) [protected], [virtual]

33.53.2 Member Function Documentation

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

Sets the range ID.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

33.53.3 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

33.54 darwin_cached_device Struct Reference

```
#include <darwin_usb.h>
```

Data Fields

- struct list_head list
- IOUSBDeviceDescriptor dev_descriptor
- UInt32 location
- UInt64 parent_session
- UInt64 session
- UInt16 address
- char sys_path [21]
- usb_device_t ** device
- int open_count
- UInt8 first_config
- UInt8 active_config
- UInt8 port
- int can_enumerate
- int refcount

33.54.1 Field Documentation

struct list_head list

IOUSBDeviceDescriptor dev_descriptor

UInt32 location

UInt64 parent_session

UInt64 session

UInt16 address

char sys_path[21]

usb_device_t device**

int open_count

UInt8 first_config

UInt8 active_config

UInt8 port

int can_enumerate

int refcount

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

- lib/ndkbuild/libusb/libusb/os/darwin_usb.h

33.55 darwin_device_handle_priv Struct Reference

```
#include <darwin_usb.h>
```

Data Structures

- struct darwin_interface

Data Fields

- int is_open
- CFRunLoopSourceRef cfSource
- struct darwin_device_handle_priv::darwin_interface interfaces [USB_MAXINTERFACES]

33.55.1 Field Documentation

int is_open

CFRunLoopSourceRef cfSource

struct darwin_device_handle_priv::darwin_interface interfaces[USB_MAXINTERFACES]

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

- lib/ndkbuild/libusb/libusb/os/darwin_usb.h

33.56 darwin_device_priv Struct Reference

```
#include <darwin_usb.h>
```

Data Fields

- struct darwin_cached_device ∗ dev

33.56.1 Field Documentation

struct darwin_cached_device∗ dev

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

- lib/ndkbuild/libusb/libusb/os/darwin_usb.h

33.57 darwin_device_handle_priv::darwin_interface Struct Reference

```
#include <darwin_usb.h>
```

Data Fields

- usb_interface_t ∗∗ interface
- uint8_t num_endpoints
- CFRunLoopSourceRef cfSource
- uint64_t frames [256]
- uint8_t endpoint_addrs [USB_MAXENDPOINTS]

33.57.1 Field Documentation

usb_interface_t∗∗ interface

uint8_t num_endpoints

CFRunLoopSourceRef cfSource

uint64_t frames[256]

uint8_t endpoint_addrs[USB_MAXENDPOINTS]

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

- lib/ndkbuild/libusb/libusb/os/darwin_usb.h

33.58 darwin_transfer_priv Struct Reference

```
#include <darwin_usb.h>
```

Data Fields

- IOUSBIsocFrame ∗ isoc_framelist
- int num_iso_packets
- IOUSBDevRequestTO req
- IOReturn result
- UInt32 size

33.58.1 Field Documentation

IOUSBIsocFrame∗ isoc_framelist

int num_iso_packets

IOUSBDevRequestTO req

IOReturn result

UInt32 size

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

- lib/ndkbuild/libusb/libusb/os/darwin_usb.h

33.59 device_priv Struct Reference

Data Fields

- char devnode [16]
- int fd
- unsigned char * cdesc
- usb_device_descriptor_t ddesc
- char * devname

33.59.1 Field Documentation

char devnode[16]

int fd

unsigned char * cdesc

usb_device_descriptor_t ddesc

char* devname

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

- lib/ndkbuild/libusb/libusb/os/netbsd_usb.c
- lib/ndkbuild/libusb/libusb/os/openbsd_usb.c

33.60 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

33.60.1 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

33.61 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.

33.61.1 Detailed Description

Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management.

33.61.2 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

33.62 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

33.62.1 Detailed Description

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

33.62.2 Constructor & Destructor Documentation

DeviceSubsystem (USBDeviceBase & *parent*) [protected]

~DeviceSubsystem ( ) [protected], [virtual]

33.62.3 Member Function Documentation

int getDeviceIndex ( ) const [protected]

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

Implemented in AnalogInputSubsystem, DigitalIOSubsystem, CounterSubsystem, DIOStreamSubsystem, and AnalogOutputSubsystem.

USBDeviceBase& getParent ( ) [inline]

Gets the parent device that this subsystem is part of.

Returns

The parent device that this subsystem is part of.

33.62.4 Field Documentation

USBDeviceBase* parent [protected]

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

- deprecated/classlib/DeviceSubsystem.hpp
- deprecated/classlib/DeviceSubsystem.cpp

33.63 devlink_cbarg Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- struct node_args * nargs
- di_node_t myself
- di_minor_t minor

33.63.1 Field Documentation

struct node_args* nargs

di_node_t myself

di_minor_t minor

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.64 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

33.64.1 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())*.

33.64.2 Constructor & Destructor Documentation

DigitalIOSubsystem (USBDeviceBase & *parent*)

~DigitalIOSubsystem ( ) [virtual]

33.64.3 Member Function Documentation

UCharArray & bitsToBytes (UCharArray & *dest,* int *bit,* const BoolArray & *src*)

BoolArray & bytesToBits (BoolArray & *dest,* const UCharArray & *src,* int *bit*)

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

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumPorts ( ) const [inline]

Gets the number of digital I/O ports (bytes).

Returns

Number of ports, numbered 0 to n-1.

int getNumChannels ( ) const [inline]

Gets the number of digital I/O channels (bits).

The number of "channels" is simply equal to the number of ports times the number of channels per port, which is eight.

Returns

Number of channels, numbered 0 to n-1.

int getNumTristateGroups ( ) const [inline]

Gets the number of tristate groups (bytes).

Returns

Number of tristate groups, numbered 0 to n-1.

int getNumTristateChannels ( ) const [inline]

Gets the number of tristate channels (bits).

The number of "channels" is simply equal to the number of tristate groups times the number of channels per group, which is eight.

Returns

Number of tristate channels, numbered 0 to n-1.

DigitalIOSubsystem & configure (bool *tristate,* const BoolArray & *outputs,* const BoolArray & *values*)

Configures the digital I/O ports.

Parameters

tristate	*true* causes all bits on the device to enter tristate (high-impedance) mode; *false* removes tristate mode. All devices with this feature power up in tristate mode, and tristate mode is changed after the remainder of the configuration has occurred.
outputs	an array of boolean values, one per digital I/O *port*. Each *true* value in the array configures the entire corresponding I/O port as an output port; each *false* value configures the entire corresponding I/O port as an input port.
values	an array of boolean values, one per digital I/O *bit*, starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each *true* value in the array sets the corresponding output bit to a "1"; each *false* value sets the corresponding output bit to a "0." The values are written to the digital output ports before the ports are taken out of tristate mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

DigitalIOSubsystem & configure (const BoolArray & *tristates,* const BoolArray & *outputs,* const BoolArray & *values*)

Configures the digital I/O ports.

If the device does not support the per-port tristate feature, then configure(bool tristate, const BoolArray &outputs, const BoolArray &values) should be used instead, otherwise an exception will be thrown.

Parameters

tristates	an array of boolean values, one per tristate group. Each *true* value in the array puts the entire corresponding I/O port into tristate (high-impedance) mode; each *false* value takes the entire corresponding I/O port out of tristate mode. All devices with this feature power up in tristate mode, and tristate mode is changed after the remainder of the configuration has occurred.
outputs	an array of boolean values, one per digital I/O *port*. Each *true* value in the array configures the entire corresponding I/O port as an output port; each *false* value configures the entire corresponding I/O port as an input port.
values	an array of boolean values, one per digital I/O *bit*, starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each *true* value in the array sets the corresponding output bit to a "1"; each *false* value sets the corresponding output bit to a "0." The values are written to the digital output ports before the ports are taken out of tristate mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

DigitalIOSubsystem & getConfiguration (BoolArray * *tristates,* BoolArray * *outputs*)

Gets the current configuration of the digital I/O ports.

If the device does not support the per-port tristate feature, then this method should not be used, otherwise an exception will be thrown.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

bool read (int *channel*)

Reads a single digital input channel.

Parameters

channel	the channel to read.

Returns

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

Exceptions

IllegalArgumentException	
OperationFailedException	

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

Reads multiple digital input channels.

Parameters

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

Returns

An array containing the values read from the specified channels. For each channel, *true* indicates that the bit is set ("1"); *false* indicates that the bit is clear ("0").

Exceptions

IllegalArgumentException	
OperationFailedException	

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

Writes a single digital output channel.

Parameters

channel	the channel to write.
value	the value to write to the specified channel. *True* sets the output bit to a "1" and *false* clears the output bit to a "0".

Returns

This subsystem, useful for chaining together multiple operations.

DigitalIOSubsystem & write (int *startChannel,* const BoolArray & *values*)

Writes multiple digital output channels.

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

33.64.4 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]`

33.64.5 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

33.65 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.

33.65.1 Detailed Description

DIOBuf: A Smart structure for maintaining bit vectors and for providing human-readable functionality to make it easy to operate on said bit vectors.

The functionality provided by this structure makes it easy for a user to work with a binary string data type convert it between raw bytes and hexidecimal representations as well as use it for generating digital intput, output and tristate bits with the corresponding `ACCES I/O Products` USB Digital input and output boards.

There are methods to work with DIOBuf that will convert this structure to a character string of 1's and 0's, to a hexadecimal representation and to raw bytes that can be used in the transmission across a number of media. This later functionality would be useful in case you are working with a network server that would need to write an incoming byte stream to a digital buffer.

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

The functionality provided by this structure makes it easy for a user to work with a binary string data type convert it between raw bytes and hexidecimal representations as well as use it for generating digital intput, output and tristate bits with the corresponding `ACCES I/O Products` USB Digital input and output boards.

There are methods to work with DIOBuf that will convert this structure to a character string of 1's and 0's, to a hexadecimal representation and to raw bytes that can be used in the transmission across a number of media. This later functionality would be useful in case you are working with a network server that would need to write an incoming byte stream to a digital buffer.

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

33.65.2 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

33.66 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

33.66.1 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())*.

33.66.2 Constructor & Destructor Documentation

DIOStreamSubsystem (USBDeviceBase & *parent*) `[protected]`

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

33.66.3 Member Function Documentation

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

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getStreamingBlockSize ( ) const `[inline]`

Gets the current streaming block size.

Returns

The current streaming block size. The value returned may not be the same as the value passed to *setStreamingBlockSize(int blockSize)* because that value is rounded up to a whole multiple of 256.

Exceptions

OperationFailedException	

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

Sets the streaming block size.

Parameters

blockSize	the streaming block size you wish to set. This will be rounded up to the next multiple of 256.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double getClock ( ) const `[inline]`

Gets the current internal read/write clock speed of a digital I/O stream.

Returns

The actual frequency that will be generated, based on the last call to *setClock(bool directionRead, double clockHz)*.

double setClock (bool *directionRead*, double *clockHz*)

Sets the internal read/write clock speed of a digital I/O stream *(see getClock())*.

Only one clock - the read or write clock

- may be active at a time, so this method ***automatically turns off*** the clock not being set by this method. Therefore, do not call this method to explicitly turn off one of the clocks because it will turn off both of them. Also, when streaming between two devices, only one should have an active internal clock; the other should have its clocks turned off *(see stopClock())*.

  Parameters

directionRead	*true* sets read clock; *false* sets write clock.
clockHz	the frequency at which to stream the samples (in Hertz).

  Returns

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

  Exceptions

OperationFailedException	

DIOStreamSubsystem& stopClock ( ) `[inline]`

Stops the internal read/write clocks of a digital I/O stream.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

DIOStreamSubsystem & open (bool *directionRead*)

Opens a digital I/O stream.

When you are done using the stream, you must close it by calling *close()*.

Parameters

directionRead	*true* open the stream for reading; *false* open the stream for writing.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

DIOStreamSubsystem & close ( )

Closes a digital I/O stream opened by a call to *open(bool directionRead)*.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

UShortArray read (int *numSamples*)

Reads a frame from a digital I/O stream opened by a call to *open(true)*.

You cannot read from, and write to a stream. A stream may be read-only or write-only.

Parameters

numSamples	the number of samples to read.

Returns

An array containing the samples read. The array may be smaller than the number of samples requested if fewer samples were received than were requested.

Exceptions

IllegalArgumentException	
OperationFailedException	

int write (const UShortArray & *values*)

Writes a frame to a digital I/O stream opened by a call to *open(false)*.

You cannot read from, and write to a stream. A stream may be read-only or write-only.

Parameters

values	an array containing the samples to write.

Returns

The number of samples actually written.

Exceptions

IllegalArgumentException	
OperationFailedException	

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

33.66.4 Friends And Related Function Documentation

friend class USB_DIO_16_Family `[friend]`

33.66.5 Field Documentation

double clockHz [protected]

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

- deprecated/classlib/DIOStreamSubsystem.hpp
- deprecated/classlib/DIOStreamSubsystem.cpp

33.67 discovered_devs Struct Reference

```
#include <libusbi.h>
```

Data Fields

- size_t len
- size_t capacity
- struct libusb_device * devices [0]

33.67.1 Field Documentation

size_t len

size_t capacity

struct libusb_device* devices[0]

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.68 DoubleArray Class Reference

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

Public Member Functions

- DoubleArray (int size=0)

33.68.1 Constructor & Destructor Documentation

DoubleArray (int *size* = 0) [inline]

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

- deprecated/classlib/USBDeviceBase.hpp

33.69 driver_lookup Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- char list [MAX_KEY_LENGTH+1]
- const DWORD reg_prop
- const char * designation

33.69.1 Field Documentation

char list[MAX_KEY_LENGTH+1]

const DWORD reg_prop

const char* designation

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.70 endpoint Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- int datafd
- int statfd

33.70.1 Field Documentation

int datafd

int statfd

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.71 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

33.71.1 Constructor & Destructor Documentation

Error ( ) [explicit]

Error (const char * *entry* **) [inline]**

33.71.2 Member Function Documentation

virtual const char* what ( ) const throw) [inline], [virtual]

33.71.3 Field Documentation

const char* message [private]

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

- samples/TestLib/TestCaseSetup.h

33.72 fx_known_device Struct Reference

```
#include <ezusb.h>
```

Data Fields

- uint16_t vid
- uint16_t pid
- int type
- const char * designation

33.72.1 Field Documentation

uint16_t vid

uint16_t pid

int type

const char* designation

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

- lib/ndkbuild/libusb/examples/ezusb.h

33.73 handle_priv Struct Reference

Data Fields

- int endpoints [USB_MAX_ENDPOINTS]

33.73.1 Field Documentation

int endpoints

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

- lib/ndkbuild/libusb/libusb/os/netbsd_usb.c
- lib/ndkbuild/libusb/libusb/os/openbsd_usb.c

33.74 hid_device_priv Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- uint16_t vid
- uint16_t pid
- uint8_t config
- uint8_t nb_interfaces
- bool uses_report_ids [3]
- uint16_t input_report_size
- uint16_t output_report_size
- uint16_t feature_report_size
- WCHAR string [3][MAX_USB_STRING_LENGTH]
- uint8_t string_index [3]

33.74.1 Field Documentation

uint16_t vid

uint16_t pid

uint8_t config

uint8_t nb_interfaces

bool uses_report_ids[3]

uint16_t input_report_size

uint16_t output_report_size

uint16_t feature_report_size

WCHAR string[3][MAX_USB_STRING_LENGTH]

uint8_t string_index[3]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.75 HIDD_ATTRIBUTES Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG Size
- USHORT VendorID
- USHORT ProductID
- USHORT VersionNumber

33.75.1 Field Documentation

ULONG Size

USHORT VendorID

USHORT ProductID

USHORT VersionNumber

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.76 HIDP_CAPS Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USAGE Usage
- USAGE UsagePage
- USHORT InputReportByteLength
- USHORT OutputReportByteLength
- USHORT FeatureReportByteLength
- USHORT Reserved [17]
- USHORT NumberLinkCollectionNodes

- USHORT NumberInputButtonCaps
- USHORT NumberInputValueCaps
- USHORT NumberInputDataIndices
- USHORT NumberOutputButtonCaps
- USHORT NumberOutputValueCaps
- USHORT NumberOutputDataIndices
- USHORT NumberFeatureButtonCaps
- USHORT NumberFeatureValueCaps
- USHORT NumberFeatureDataIndices

33.76.1 Field Documentation

USAGE Usage

USAGE UsagePage

USHORT InputReportByteLength

USHORT OutputReportByteLength

USHORT FeatureReportByteLength

USHORT Reserved[17]

USHORT NumberLinkCollectionNodes

USHORT NumberInputButtonCaps

USHORT NumberInputValueCaps

USHORT NumberInputDataIndices

USHORT NumberOutputButtonCaps

USHORT NumberOutputValueCaps

USHORT NumberOutputDataIndices

USHORT NumberFeatureButtonCaps

USHORT NumberFeatureValueCaps

USHORT NumberFeatureDataIndices

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.77 htab_entry Struct Reference

Data Fields

- unsigned long used
- char * str

33.77.1 Field Documentation

unsigned long used

char* str

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

- lib/ndkbuild/libusb/libusb/os/windows_nt_common.c

33.78 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.

33.78.1 Detailed Description

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

33.78.2 Constructor & Destructor Documentation

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

Constructs the exception from a simple string message.

Parameters

message	The text of the message.

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

- deprecated/classlib/USBDeviceManager.hpp

33.79 imaxdiv_t Struct Reference

This file has no copyright assigned and is placed in the Public Domain.

`#include <inttypes.h>`

Data Fields

- intmax_t quot
- intmax_t rem

33.79.1 Detailed Description

This file has no copyright assigned and is placed in the Public Domain.

This file was original part of the w64 mingw-runtime package.

33.79.2 Field Documentation

intmax_t quot

intmax_t rem

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

- lib/ndkbuild/libusb/msvc/inttypes.h

33.80 IntArray Class Reference

`#include <USBDeviceBase.hpp>`

Public Member Functions

- IntArray (int size=0)

33.80.1 Constructor & Destructor Documentation

IntArray (int *size* = 0) [inline]

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

- deprecated/classlib/USBDeviceBase.hpp

33.81 interface_handle_t Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- HANDLE dev_handle
- HANDLE api_handle

33.81.1 Field Documentation

HANDLE dev_handle

HANDLE api_handle

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.82 libusb_bos_descriptor Struct Reference

A structure representing the Binary Device Object Store (BOS) descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint16_t wTotalLength

  Length of this descriptor and all of its sub descriptors.
- uint8_t bNumDeviceCaps

  The number of separate device capability descriptors in the BOS.
- struct libusb_bos_dev_capability_descriptor * dev_capability [0]

  bNumDeviceCap Device Capability Descriptors

33.82.1 Detailed Description

A structure representing the Binary Device Object Store (BOS) descriptor.

This descriptor is documented in section 9.6.2 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.82.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_BOS LIBUSB_DT_BOS in this context.

uint16_t wTotalLength

Length of this descriptor and all of its sub descriptors.

uint8_t bNumDeviceCaps

The number of separate device capability descriptors in the BOS.

struct libusb_bos_dev_capability_descriptor* dev_capability[0]

bNumDeviceCap Device Capability Descriptors

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.83 libusb_bos_dev_capability_descriptor Struct Reference

A generic representation of a BOS Device Capability descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint8_t bDevCapabilityType

  Device Capability type.
- uint8_t dev_capability_data [0]

  Device Capability data (bLength - 3 bytes)

33.83.1 Detailed Description

A generic representation of a BOS Device Capability descriptor.

It is advised to check bDevCapabilityType and call the matching libusb_get_*_descriptor function to get a structure fully matching the type.

33.83.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_DEVICE_CAPABILITY LIBUSB_DT_DEVICE_CAPABILITY in this context.

uint8_t bDevCapabilityType

Device Capability type.

uint8_t dev_capability_data[0]

Device Capability data (bLength - 3 bytes)

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.84 libusb_config_descriptor Struct Reference

A structure representing the standard USB configuration descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength
  Size of this descriptor (in bytes)
- uint8_t bDescriptorType
  Descriptor type.
- uint16_t wTotalLength
  Total length of data returned for this configuration.
- uint8_t bNumInterfaces
  Number of interfaces supported by this configuration.
- uint8_t bConfigurationValue
  Identifier value for this configuration.
- uint8_t iConfiguration
  Index of string descriptor describing this configuration.
- uint8_t bmAttributes
  Configuration characteristics.
- uint8_t MaxPower
  Maximum power consumption of the USB device from this bus in this configuration when the device is fully operation.
- const struct libusb_interface * interface
  Array of interfaces supported by this configuration.
- const unsigned char * extra
  Extra descriptors.
- int extra_length
  Length of the extra descriptors, in bytes.

33.84.1 Detailed Description

A structure representing the standard USB configuration descriptor.

This descriptor is documented in section 9.6.3 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.84.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_CONFIG LIBUSB_DT_CONFIG in this context.

uint16_t wTotalLength

Total length of data returned for this configuration.

uint8_t bNumInterfaces

Number of interfaces supported by this configuration.

uint8_t bConfigurationValue

Identifier value for this configuration.

uint8_t iConfiguration

Index of string descriptor describing this configuration.

uint8_t bmAttributes

Configuration characteristics.

uint8_t MaxPower

Maximum power consumption of the USB device from this bus in this configuration when the device is fully operation.

Expressed in units of 2 mA when the device is operating in high-speed mode and in units of 8 mA when the device is operating in super-speed mode.

const struct libusb_interface* interface

Array of interfaces supported by this configuration.

The length of this array is determined by the bNumInterfaces field.

const unsigned char* extra

Extra descriptors.

If libusb encounters unknown configuration descriptors, it will store them here, should you wish to parse them.

int extra_length

Length of the extra descriptors, in bytes.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.85 libusb_container_id_descriptor Struct Reference

A structure representing the Container ID descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength
  Size of this descriptor (in bytes)
- uint8_t bDescriptorType
  Descriptor type.
- uint8_t bDevCapabilityType
  Capability type.
- uint8_t bReserved
  Reserved field.
- uint8_t ContainerID [16]
  128 bit UUID

33.85.1 Detailed Description

A structure representing the Container ID descriptor.

This descriptor is documented in section 9.6.2.3 of the USB 3.0 specification. All multiple-byte fields, except UUIDs, are represented in host-endian format.

33.85.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_DEVICE_CAPABILITY LIBUSB_DT_DEVICE_CAPABILITY in this context.

uint8_t bDevCapabilityType

Capability type.

Will have value libusb_capability_type::LIBUSB_BT_CONTAINER_ID LIBUSB_BT_CONTAINER_ID in this context.

uint8_t bReserved

Reserved field.

uint8_t ContainerID[16]

128 bit UUID

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.86 libusb_context Struct Reference

```
#include <libusbi.h>
```

Data Fields

- int debug
- int debug_fixed
- int event_pipe [2]
- struct list_head usb_devs
- usbi_mutex_t usb_devs_lock
- struct list_head open_devs
- usbi_mutex_t open_devs_lock
- struct list_head hotplug_cbs
- usbi_mutex_t hotplug_cbs_lock
- struct list_head flying_transfers
- usbi_mutex_t flying_transfers_lock
- libusb_pollfd_added_cb fd_added_cb
- libusb_pollfd_removed_cb fd_removed_cb
- void * fd_cb_user_data
- usbi_mutex_t events_lock
- int event_handler_active
- usbi_tls_key_t event_handling_key
- usbi_mutex_t event_waiters_lock
- usbi_cond_t event_waiters_cond
- usbi_mutex_t event_data_lock
- unsigned int event_flags
- unsigned int device_close
- struct list_head ipollfds
- struct pollfd * pollfds
- POLL_NFDS_TYPE pollfds_cnt
- struct list_head hotplug_msgs
- struct list_head completed_transfers
- struct list_head list

33.86.1 Field Documentation

int debug

int debug_fixed

int event_pipe[2]

struct list_head usb_devs

usbi_mutex_t usb_devs_lock

struct list_head open_devs

usbi_mutex_t open_devs_lock

struct list_head hotplug_cbs

usbi_mutex_t hotplug_cbs_lock

struct list_head flying_transfers

usbi_mutex_t flying_transfers_lock

libusb_pollfd_added_cb fd_added_cb

libusb_pollfd_removed_cb fd_removed_cb

void* fd_cb_user_data

usbi_mutex_t events_lock

int event_handler_active

usbi_tls_key_t event_handling_key

usbi_mutex_t event_waiters_lock

usbi_cond_t event_waiters_cond

usbi_mutex_t event_data_lock

unsigned int event_flags

unsigned int device_close

struct list_head ipollfds

struct pollfd* pollfds

POLL_NFDS_TYPE pollfds_cnt

struct list_head hotplug_msgs

struct list_head completed_transfers

struct list_head list

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.87 libusb_control_setup Struct Reference

Setup packet for control transfers.

```
#include <libusb.h>
```

Data Fields

- uint8_t bmRequestType

  Request type.
- uint8_t bRequest

  Request.
- uint16_t wValue

  Value.
- uint16_t wIndex

  Index.
- uint16_t wLength

  Number of bytes to transfer.

33.87.1 Detailed Description

Setup packet for control transfers.

33.87.2 Field Documentation

uint8_t bmRequestType

Request type.

Bits 0:4 determine recipient, see libusb_request_recipient. Bits 5:6 determine type, see libusb_request_type. Bit 7 determines data transfer direction, see libusb_endpoint_direction.

uint8_t bRequest

Request.

If the type bits of bmRequestType are equal to LIBUSB_REQUEST_TYPE_STANDARD then this field refers to libusb_standard_request. For other cases, use of this field is application-specific.

uint16_t wValue

Value.

Varies according to request

uint16_t wIndex

Index.

Varies according to request, typically used to pass an index or offset

uint16_t wLength

Number of bytes to transfer.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.88 libusb_device Struct Reference

```
#include <libusbi.h>
```

Data Fields

- usbi_mutex_t lock
- int refcnt
- struct libusb_context * ctx
- uint8_t bus_number
- uint8_t port_number
- struct libusb_device * parent_dev

- uint8_t device_address
- uint8_t num_configurations
- enum libusb_speed speed
- struct list_head list
- unsigned long session_data
- struct libusb_device_descriptor device_descriptor
- int attached
- unsigned char os_priv [0]

33.88.1 Field Documentation

usbi_mutex_t lock

int refcnt

struct libusb_context* ctx

uint8_t bus_number

uint8_t port_number

struct libusb_device* parent_dev

uint8_t device_address

uint8_t num_configurations

enum libusb_speed speed

struct list_head list

unsigned long session_data

struct libusb_device_descriptor device_descriptor

int attached

unsigned char os_priv[0]

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.89 libusb_device_descriptor Struct Reference

A structure representing the standard USB device descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint16_t bcdUSB

  USB specification release number in binary-coded decimal.
- uint8_t bDeviceClass

  USB-IF class code for the device.
- uint8_t bDeviceSubClass

  USB-IF subclass code for the device, qualified by the bDeviceClass value.
- uint8_t bDeviceProtocol

  USB-IF protocol code for the device, qualified by the bDeviceClass and bDeviceSubClass values.
- uint8_t bMaxPacketSize0

  Maximum packet size for endpoint 0.
- uint16_t idVendor

USB-IF vendor ID.

- uint16_t idProduct

  USB-IF product ID.
- uint16_t bcdDevice

  Device release number in binary-coded decimal.
- uint8_t iManufacturer

  Index of string descriptor describing manufacturer.
- uint8_t iProduct

  Index of string descriptor describing product.
- uint8_t iSerialNumber

  Index of string descriptor containing device serial number.
- uint8_t bNumConfigurations

  Number of possible configurations.

33.89.1 Detailed Description

A structure representing the standard USB device descriptor.

This descriptor is documented in section 9.6.1 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.89.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_DEVICE LIBUSB_DT_DEVICE in this context.

uint16_t bcdUSB

USB specification release number in binary-coded decimal.

A value of 0x0200 indicates USB 2.0, 0x0110 indicates USB 1.1, etc.

uint8_t bDeviceClass

USB-IF class code for the device.

See libusb_class_code.

uint8_t bDeviceSubClass

USB-IF subclass code for the device, qualified by the bDeviceClass value.

uint8_t bDeviceProtocol

USB-IF protocol code for the device, qualified by the bDeviceClass and bDeviceSubClass values.

uint8_t bMaxPacketSize0

Maximum packet size for endpoint 0.

uint16_t idVendor

USB-IF vendor ID.

uint16_t idProduct

USB-IF product ID.

uint16_t bcdDevice

Device release number in binary-coded decimal.

uint8_t iManufacturer

Index of string descriptor describing manufacturer.

uint8_t iProduct

Index of string descriptor describing product.

uint8_t iSerialNumber

Index of string descriptor containing device serial number.

uint8_t bNumConfigurations

Number of possible configurations.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.90 libusb_device_handle Struct Reference

```
#include <libusbi.h>
```

Data Fields

- usbi_mutex_t lock
- unsigned long claimed_interfaces
- struct list_head list
- struct libusb_device * dev
- int auto_detach_kernel_driver
- unsigned char os_priv [0]

33.90.1 Field Documentation

usbi_mutex_t lock

unsigned long claimed_interfaces

struct list_head list

struct libusb_device* dev

int auto_detach_kernel_driver

unsigned char os_priv[0]

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.91 libusb_endpoint_descriptor Struct Reference

A structure representing the standard USB endpoint descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint8_t bEndpointAddress

  The address of the endpoint described by this descriptor.
- uint8_t bmAttributes

  Attributes which apply to the endpoint when it is configured using the bConfigurationValue.
- uint16_t wMaxPacketSize

  Maximum packet size this endpoint is capable of sending/receiving.
- uint8_t bInterval

  Interval for polling endpoint for data transfers.
- uint8_t bRefresh

  For audio devices only: the rate at which synchronization feedback is provided.
- uint8_t bSynchAddress

  For audio devices only: the address if the synch endpoint.
- const unsigned char * extra

  Extra descriptors.
- int extra_length

  Length of the extra descriptors, in bytes.

33.91.1 Detailed Description

A structure representing the standard USB endpoint descriptor.

This descriptor is documented in section 9.6.6 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.91.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_ENDPOINT LIBUSB_DT_ENDPOINT in this context.

uint8_t bEndpointAddress

The address of the endpoint described by this descriptor.

Bits 0:3 are the endpoint number. Bits 4:6 are reserved. Bit 7 indicates direction, see libusb_endpoint_direction.

uint8_t bmAttributes

Attributes which apply to the endpoint when it is configured using the bConfigurationValue.

Bits 0:1 determine the transfer type and correspond to libusb_transfer_type. Bits 2:3 are only used for isochronous endpoints and correspond to libusb_iso_sync_type. Bits 4:5 are also only used for isochronous endpoints and correspond to libusb_iso_usage_type. Bits 6:7 are reserved.

uint16_t wMaxPacketSize

Maximum packet size this endpoint is capable of sending/receiving.

uint8_t bInterval

Interval for polling endpoint for data transfers.

uint8_t bRefresh

For audio devices only: the rate at which synchronization feedback is provided.

uint8_t bSynchAddress

For audio devices only: the address if the synch endpoint.

const unsigned char* extra

Extra descriptors.

If libusb encounters unknown endpoint descriptors, it will store them here, should you wish to parse them.

int extra_length

Length of the extra descriptors, in bytes.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.92 libusb_hid_descriptor Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint16_t bcdHID
- uint8_t bCountryCode
- uint8_t bNumDescriptors
- uint8_t bClassDescriptorType
- uint16_t wClassDescriptorLength

33.92.1 Field Documentation

uint8_t bLength

uint8_t bDescriptorType

uint16_t bcdHID

uint8_t bCountryCode

uint8_t bNumDescriptors

uint8_t bClassDescriptorType

uint16_t wClassDescriptorLength

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.93 libusb_hotplug_callback Struct Reference

The hotplug callback structure.

```
#include <hotplug.h>
```

Data Fields

- struct libusb_context * ctx

  Context this callback is associated with.
- int vendor_id

  Vendor ID to match or LIBUSB_HOTPLUG_MATCH_ANY.
- int product_id

  Product ID to match or LIBUSB_HOTPLUG_MATCH_ANY.
- int dev_class

  Device class to match or LIBUSB_HOTPLUG_MATCH_ANY.
- libusb_hotplug_flag flags

  Hotplug callback flags.
- libusb_hotplug_event events

  Event(s) that will trigger this callback.
- libusb_hotplug_callback_fn cb

  Callback function to invoke for matching event/device.
- libusb_hotplug_callback_handle handle

  Handle for this callback (used to match on deregister)
- void * user_data

  User data that will be passed to the callback function.
- int needs_free

  Callback is marked for deletion.
- struct list_head list

  List this callback is registered in (ctx->hotplug_cbs)

33.93.1 Detailed Description

The hotplug callback structure.

The user populates this structure with libusb_hotplug_prepare_callback() and then calls libusb_hotplug_register_callback() to receive notification of hotplug events.

33.93.2 Field Documentation

struct libusb_context* ctx

Context this callback is associated with.

int vendor_id

Vendor ID to match or LIBUSB_HOTPLUG_MATCH_ANY.

int product_id

Product ID to match or LIBUSB_HOTPLUG_MATCH_ANY.

int dev_class

Device class to match or LIBUSB_HOTPLUG_MATCH_ANY.

libusb_hotplug_flag flags

Hotplug callback flags.

libusb_hotplug_event events

Event(s) that will trigger this callback.

libusb_hotplug_callback_fn cb

Callback function to invoke for matching event/device.

libusb_hotplug_callback_handle handle

Handle for this callback (used to match on deregister)

void* user_data

User data that will be passed to the callback function.

int needs_free

Callback is marked for deletion.

struct list_head list

List this callback is registered in (ctx->hotplug_cbs)

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

- lib/ndkbuild/libusb/libusb/hotplug.h

33.94 libusb_hotplug_message Struct Reference

```
#include <hotplug.h>
```

Data Fields

- libusb_hotplug_event event
  The hotplug event that occurred.
- struct libusb_device * device
  The device for which this hotplug event occurred.
- struct list_head list
  List this message is contained in (ctx->hotplug_msgs)

33.94.1 Field Documentation

libusb_hotplug_event event

The hotplug event that occurred.

struct libusb_device* device

The device for which this hotplug event occurred.

struct list_head list

List this message is contained in (ctx->hotplug_msgs)

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

- lib/ndkbuild/libusb/libusb/hotplug.h

33.95 libusb_interface Struct Reference

A collection of alternate settings for a particular USB interface.

```
#include <libusb.h>
```

Data Fields

- const struct libusb_interface_descriptor * altsetting
  Array of interface descriptors.
- int num_altsetting
  The number of alternate settings that belong to this interface.

33.95.1 Detailed Description

A collection of alternate settings for a particular USB interface.

33.95.2 Field Documentation

const struct libusb_interface_descriptor* altsetting

Array of interface descriptors.

The length of this array is determined by the num_altsetting field.

int num_altsetting

The number of alternate settings that belong to this interface.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.96 libusb_interface_descriptor Struct Reference

A structure representing the standard USB interface descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength
  Size of this descriptor (in bytes)
- uint8_t bDescriptorType
  Descriptor type.
- uint8_t bInterfaceNumber
  Number of this interface.
- uint8_t bAlternateSetting
  Value used to select this alternate setting for this interface.
- uint8_t bNumEndpoints
  Number of endpoints used by this interface (excluding the control endpoint).
- uint8_t bInterfaceClass
  USB-IF class code for this interface.
- uint8_t bInterfaceSubClass
  USB-IF subclass code for this interface, qualified by the bInterfaceClass value.
- uint8_t bInterfaceProtocol
  USB-IF protocol code for this interface, qualified by the bInterfaceClass and bInterfaceSubClass values.
- uint8_t iInterface
  Index of string descriptor describing this interface.
- const struct libusb_endpoint_descriptor * endpoint
  Array of endpoint descriptors.
- const unsigned char * extra
  Extra descriptors.
- int extra_length
  Length of the extra descriptors, in bytes.

33.96.1 Detailed Description

A structure representing the standard USB interface descriptor.

This descriptor is documented in section 9.6.5 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.96.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_INTERFACE LIBUSB_DT_INTERFACE in this context.

uint8_t bInterfaceNumber

Number of this interface.

uint8_t bAlternateSetting

Value used to select this alternate setting for this interface.

uint8_t bNumEndpoints

Number of endpoints used by this interface (excluding the control endpoint).

uint8_t bInterfaceClass

USB-IF class code for this interface.

See libusb_class_code.

uint8_t bInterfaceSubClass

USB-IF subclass code for this interface, qualified by the bInterfaceClass value.

uint8_t bInterfaceProtocol

USB-IF protocol code for this interface, qualified by the bInterfaceClass and bInterfaceSubClass values.

uint8_t iInterface

Index of string descriptor describing this interface.

const struct libusb_endpoint_descriptor* endpoint

Array of endpoint descriptors.

This length of this array is determined by the bNumEndpoints field.

const unsigned char* extra

Extra descriptors.

If libusb encounters unknown interface descriptors, it will store them here, should you wish to parse them.

int extra_length

Length of the extra descriptors, in bytes.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.97 libusb_iso_packet_descriptor Struct Reference

Isochronous packet descriptor.

```
#include <libusb.h>
```

Data Fields

- unsigned int length

  Length of data to request in this packet.
- unsigned int actual_length

  Amount of data that was actually transferred.
- enum libusb_transfer_status status

  Status code for this packet.

33.97.1 Detailed Description

Isochronous packet descriptor.

33.97.2 Field Documentation

unsigned int length

Length of data to request in this packet.

unsigned int actual_length

Amount of data that was actually transferred.

enum libusb_transfer_status status

Status code for this packet.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.98 libusb_pollfd Struct Reference

File descriptor for polling.

```
#include <libusb.h>
```

Data Fields

- int fd

  Numeric file descriptor.
- short events

  Event flags to poll for from <poll.h>.

33.98.1 Detailed Description

File descriptor for polling.

33.98.2 Field Documentation

int fd

Numeric file descriptor.

short events

Event flags to poll for from <poll.h>.

POLLIN indicates that you should monitor this file descriptor for becoming ready to read from, and POLLOUT indicates that you should monitor this file descriptor for nonblocking write readiness.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.99 libusb_ss_endpoint_companion_descriptor Struct Reference

A structure representing the superspeed endpoint companion descriptor.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint8_t bMaxBurst

  The maximum number of packets the endpoint can send or receive as part of a burst.
- uint8_t bmAttributes

  In bulk EP: bits 4:0 represents the maximum number of streams the EP supports.
- uint16_t wBytesPerInterval

  The total number of bytes this EP will transfer every service interval.

33.99.1 Detailed Description

A structure representing the superspeed endpoint companion descriptor.

This descriptor is documented in section 9.6.7 of the USB 3.0 specification. All multiple-byte fields are represented in host-endian format.

33.99.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_SS_ENDPOINT_COMPANION in this context.

uint8_t bMaxBurst

The maximum number of packets the endpoint can send or receive as part of a burst.

uint8_t bmAttributes

In bulk EP: bits 4:0 represents the maximum number of streams the EP supports.

In isochronous EP: bits 1:0 represents the Mult - a zero based value that determines the maximum number of packets within a service interval

uint16_t wBytesPerInterval

The total number of bytes this EP will transfer every service interval.

valid only for periodic EPs.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.100 libusb_ss_usb_device_capability_descriptor Struct Reference

A structure representing the SuperSpeed USB Device Capability descriptor This descriptor is documented in section 9.6.2.2 of the USB 3.0 specification.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength

  Size of this descriptor (in bytes)
- uint8_t bDescriptorType

  Descriptor type.
- uint8_t bDevCapabilityType

  Capability type.
- uint8_t bmAttributes

  Bitmap encoding of supported device level features.
- uint16_t wSpeedSupported

  Bitmap encoding of the speed supported by this device when operating in SuperSpeed mode.
- uint8_t bFunctionalitySupport

  The lowest speed at which all the functionality supported by the device is available to the user.
- uint8_t bU1DevExitLat

  U1 Device Exit Latency.
- uint16_t bU2DevExitLat

  U2 Device Exit Latency.

33.100.1 Detailed Description

A structure representing the SuperSpeed USB Device Capability descriptor This descriptor is documented in section 9.6.2.2 of the USB 3.0 specification.

All multiple-byte fields are represented in host-endian format.

33.100.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_DEVICE_CAPABILITY LIBUSB_DT_DEVICE_CAPABILITY in this context.

uint8_t bDevCapabilityType

Capability type.

Will have value libusb_capability_type::LIBUSB_BT_SS_USB_DEVICE_CAPABILITY LIBUSB_BT_SS_USB_DEVICE-_CAPABILITY in this context.

uint8_t bmAttributes

Bitmap encoding of supported device level features.

A value of one in a bit location indicates a feature is supported; a value of zero indicates it is not supported. See libusb_ss_usb_device_capability_attributes.

uint16_t wSpeedSupported

Bitmap encoding of the speed supported by this device when operating in SuperSpeed mode.

See libusb_supported_speed.

uint8_t bFunctionalitySupport

The lowest speed at which all the functionality supported by the device is available to the user.

For example if the device supports all its functionality when connected at full speed and above then it sets this value to 1.

uint8_t bU1DevExitLat

U1 Device Exit Latency.

uint16_t bU2DevExitLat

U2 Device Exit Latency.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.101 libusb_testlib_ctx Struct Reference

Context for test library functions.

```
#include <libusb_testlib.h>
```

Data Fields

- char ** test_names
- int test_count
- bool list_tests
- bool verbose
- int old_stdout
- int old_stderr
- FILE * output_file
- int null_fd

33.101.1 Detailed Description

Context for test library functions.

33.101.2 Field Documentation

char test_names**

int test_count

bool list_tests

bool verbose

int old_stdout

int old_stderr

FILE* output_file

int null_fd

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

- lib/ndkbuild/libusb/tests/libusb_testlib.h

33.102 libusb_testlib_test Struct Reference

Structure holding a test description.

```
#include <libusb_testlib.h>
```

Data Fields

- const char * name

  Human readable name of the test.
- libusb_testlib_test_function function

  The test library will call this function to run the test.

33.102.1 Detailed Description

Structure holding a test description.

33.102.2 Field Documentation

const char* name

Human readable name of the test.

libusb_testlib_test_function function

The test library will call this function to run the test.

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

- lib/ndkbuild/libusb/tests/libusb_testlib.h

33.103 libusb_transfer Struct Reference

The generic USB transfer structure.

```
#include <libusb.h>
```

Data Fields

- libusb_device_handle * dev_handle

  Handle of the device that this transfer will be submitted to.
- uint8_t flags

  A bitwise OR combination of libusb_transfer_flags.
- unsigned char endpoint

  Address of the endpoint where this transfer will be sent.
- unsigned char type

  Type of the endpoint from libusb_transfer_type.
- unsigned int timeout

  Timeout for this transfer in millseconds.
- enum libusb_transfer_status status

  The status of the transfer.
- int length

  Length of the data buffer.
- int actual_length

  Actual length of data that was transferred.
- libusb_transfer_cb_fn callback

  Callback function.
- void * user_data

  User context data to pass to the callback function.
- unsigned char * buffer

  Data buffer.
- int num_iso_packets

  Number of isochronous packets.
- struct libusb_iso_packet_descriptor iso_packet_desc [0]

  Isochronous packet descriptors, for isochronous transfers only.

33.103.1 Detailed Description

The generic USB transfer structure.

The user populates this structure and then submits it in order to request a transfer. After the transfer has completed, the library populates the transfer with the results and passes it back to the user.

33.103.2 Field Documentation

libusb_device_handle* dev_handle

Handle of the device that this transfer will be submitted to.

uint8_t flags

A bitwise OR combination of libusb_transfer_flags.

unsigned char endpoint

Address of the endpoint where this transfer will be sent.

unsigned char type

Type of the endpoint from libusb_transfer_type.

unsigned int timeout

Timeout for this transfer in millseconds.

A value of 0 indicates no timeout.

enum libusb_transfer_status status

The status of the transfer.

Read-only, and only for use within transfer callback function.

If this is an isochronous transfer, this field may read COMPLETED even if there were errors in the frames. Use the status field in each packet to determine if errors occurred.

int length

Length of the data buffer.

int actual_length

Actual length of data that was transferred.

Read-only, and only for use within transfer callback function. Not valid for isochronous endpoint transfers.

libusb_transfer_cb_fn callback

Callback function.

This will be invoked when the transfer completes, fails, or is cancelled.

void* user_data

User context data to pass to the callback function.

unsigned char* buffer

Data buffer.

int num_iso_packets

Number of isochronous packets.

Only used for I/O with isochronous endpoints.

struct libusb_iso_packet_descriptor iso_packet_desc[0]

Isochronous packet descriptors, for isochronous transfers only.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.104 libusb_usb_2_0_extension_descriptor Struct Reference

A structure representing the USB 2.0 Extension descriptor This descriptor is documented in section 9.6.2.1 of the USB 3.0 specification.

```
#include <libusb.h>
```

Data Fields

- uint8_t bLength
  Size of this descriptor (in bytes)
- uint8_t bDescriptorType
  Descriptor type.
- uint8_t bDevCapabilityType
  Capability type.
- uint32_t bmAttributes
  Bitmap encoding of supported device level features.

33.104.1 Detailed Description

A structure representing the USB 2.0 Extension descriptor This descriptor is documented in section 9.6.2.1 of the USB 3.0 specification.

All multiple-byte fields are represented in host-endian format.

33.104.2 Field Documentation

uint8_t bLength

Size of this descriptor (in bytes)

uint8_t bDescriptorType

Descriptor type.

Will have value libusb_descriptor_type::LIBUSB_DT_DEVICE_CAPABILITY LIBUSB_DT_DEVICE_CAPABILITY in this context.

uint8_t bDevCapabilityType

Capability type.

Will have value libusb_capability_type::LIBUSB_BT_USB_2_0_EXTENSION LIBUSB_BT_USB_2_0_EXTENSION in this context.

uint32_t bmAttributes

Bitmap encoding of supported device level features.

A value of one in a bit location indicates a feature is supported; a value of zero indicates it is not supported. See libusb_usb_2_0_extension_attributes.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.105 libusb_version Struct Reference

Structure providing the version of the libusb runtime.

```
#include <libusb.h>
```

Data Fields

- const uint16_t major
  Library major version.
- const uint16_t minor
  Library minor version.
- const uint16_t micro
  Library micro version.
- const uint16_t nano
  Library nano version.
- const char * rc
  Library release candidate suffix string, e.g.
- const char * describe
  For ABI compatibility only.

33.105.1 Detailed Description

Structure providing the version of the libusb runtime.

33.105.2 Field Documentation

const uint16_t major

Library major version.

const uint16_t minor

Library minor version.

const uint16_t micro

Library micro version.

const uint16_t nano

Library nano version.

const char* rc

Library release candidate suffix string, e.g.

"-rc4".

const char* describe

For ABI compatibility only.

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

- lib/ndkbuild/libusb/libusb/libusb.h

33.106 linux_device_handle_priv Struct Reference

Data Fields

- int fd
- int fd_removed
- uint32_t caps

33.106.1 Field Documentation

int fd

int fd_removed

uint32_t caps

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.c

33.107 linux_device_priv Struct Reference

Data Fields

- char * sysfs_dir
- unsigned char * descriptors
- int descriptors_len
- int active_config

33.107.1 Field Documentation

char* sysfs_dir

unsigned char* descriptors

int descriptors_len

int active_config

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.c

33.108 linux_transfer_priv Struct Reference

Data Fields

- union {
  struct usbfs_urb * urbs
  struct usbfs_urb ** iso_urbs
  };

- enum reap_action reap_action
- int num_urbs
- int num_retired
- enum libusb_transfer_status reap_status
- int iso_packet_offset

33.108.1 Field Documentation

struct usbfs_urb* urbs

struct usbfs_urb iso_urbs**

union { ... }

enum reap_action reap_action

int num_urbs

int num_retired

enum libusb_transfer_status reap_status

int iso_packet_offset

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.c

33.109 list_head Struct Reference

```
#include <libusbi.h>
```

Data Fields

- struct list_head * prev
- struct list_head * next

33.109.1 Field Documentation

struct list_head* prev

struct list_head * next

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.110 lookup Struct Reference

```
#include <AIOTypes.h>
```

Data Fields

- int value
- char * str
- char * strvalue

33.110.1 Field Documentation

int value

char * str

char * strvalue

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

- lib/AIOTypes.h

33.111 mux_settings Struct Reference

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

Data Fields

- unsigned long ADCChannelsPerGroup
- unsigned long ADCMUXChannels
- AIOUSB_BOOL defined

33.111.1 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

33.112 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)

33.112.1 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

33.113 node_args Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- struct libusb_context ∗ ctx
- struct discovered_devs ∗∗ discdevs
- const char ∗ last_ugenpath
- di_devlink_handle_t dlink_hdl

33.113.1 Field Documentation

struct libusb_context∗ ctx

struct discovered_devs∗∗ discdevs

const char∗ last_ugenpath

di_devlink_handle_t dlink_hdl

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.114 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.

33.114.1 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.

33.114.2 Constructor & Destructor Documentation

OperationFailedException (int *result*) `[inline]`

Constructs the exception from an AIOUSB module error code.

Parameters

result	AIOUSB module result code.

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

Constructs the exception from a simple string message.

Parameters

message	The text of the message.

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

- deprecated/classlib/USBDeviceManager.hpp

33.115 option Struct Reference

```
#include <getopt.h>
```

Data Fields

- char * name
- int has_arg
- int * flag
- int val

33.115.1 Field Documentation

char* name

int has_arg

int* flag

int val

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

- lib/ndkbuild/libusb/examples/getopt/getopt.h

33.116 options Struct Reference

Data Fields

- int maxcount
- int use_maxcount

33.116.1 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

33.117 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

33.117.1 Field Documentation

int64_t num_scans

int64_t default_num_scans

int num_channels

int default_num_channels

int num_oversamples

int default_num_oversamples

int gain_code

int clock_rate

int default_clock_rate

char∗ outfile

int reset

int debug_level

int number_ranges

int verbose

int start_channel

int default_start_channel

int end_channel

int default_end_channel

int index

int block_size

int with_timing

int slow_acquire

int buffer_size

int rate_limit

int physical

int counts

int calibration

int repeat

char∗ aiobuf_json

char∗ default_aiobuf_json

char∗ adcconfig_json

struct channel_range∗∗ ranges

int num_scans

int clock_speed

int cal_channel

int max_channels

int clock_scale

int calibration_enabled

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

- samples/TestLib/aiocommon.h
- samples/USB-AI16-16/bulk_acquire_sample.c

33.118 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.

33.118.1 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.

33.118.2 Constructor & Destructor Documentation

OutputVoltagePoint ( ) [inline]

Default constructor for analog output data point.

OutputVoltagePoint (int *channel*, double *volts*) [inline]

Constructor for analog output data point.

Parameters

channel	the channel number to output voltage to.
volts	the voltage to output.

33.118.3 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

33.119 OutputVoltagePointArray Class Reference

```
#include <OutputVoltagePoint.hpp>
```

Public Member Functions

- OutputVoltagePointArray (int size=0)

33.119.1 Constructor & Destructor Documentation

OutputVoltagePointArray (int *size* = 0) [inline]

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

- deprecated/classlib/OutputVoltagePoint.hpp

33.120 pollfd Struct Reference

```
#include <poll_windows.h>
```

Data Fields

- int fd
- short events
- short revents

33.120.1 Field Documentation

int fd

short events

short revents

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

- lib/ndkbuild/libusb/libusb/os/poll_windows.h

33.121 ProductIDName Struct Reference

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

Data Fields

- unsigned int id
- char name [PROD_NAME_SIZE+2]

33.121.1 Field Documentation

unsigned int id

char name

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

- lib/AIOUSB_Core.h

33.122 ram_poke_context Struct Reference

Data Fields

- libusb_device_handle * device
- ram_mode mode
- size_t total
- size_t count

33.122.1 Field Documentation

libusb_device_handle* device

ram_mode mode

size_t total

size_t count

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

- lib/ndkbuild/libusb/examples/ezusb.c

33.123 rangelookup Struct Reference

Data Fields

- int minvalue
- int maxvalue

33.123.1 Field Documentation

int minvalue

int maxvalue

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

- lib/AIOContinuousBuffer.c

33.124 RosterLooper Class Reference

Public Member Functions

- RosterLooper (USBRoster *)
- void Stop ()
- virtual void MessageReceived (BMessage *)
- bool InitCheck ()

Private Attributes

- USBRoster * fRoster
- WatchedEntry * fRoot
- BMessenger * fMessenger
- bool fInitCheck

33.124.1 Constructor & Destructor Documentation

RosterLooper (USBRoster * *roster*)

33.124.2 Member Function Documentation

void Stop ( )

void MessageReceived (BMessage * *message*) `[virtual]`

bool InitCheck ( )

33.124.3 Field Documentation

USBRoster* fRoster `[private]`

WatchedEntry* fRoot `[private]`

BMessenger* fMessenger `[private]`

bool fInitCheck `[private]`

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

- lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp

33.125 StringArray Class Reference

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

Public Member Functions

- StringArray (int size=0)

33.125.1 Constructor & Destructor Documentation

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

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

- deprecated/classlib/USBDeviceBase.hpp

33.126 sunos_device_handle_priv Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- uint8_t altsetting [USB_MAXINTERFACES]
- uint8_t config_index
- sunos_ep_priv_t eps [USB_MAXENDPOINTS]
- sunos_dev_priv_t * dpriv

33.126.1 Field Documentation

uint8_t altsetting[USB_MAXINTERFACES]

uint8_t config_index

sunos_ep_priv_t eps[USB_MAXENDPOINTS]

sunos_dev_priv_t* dpriv

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.127 sunos_device_priv Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- uint8_t cfgvalue
- uint8_t * raw_cfgdescr
- struct libusb_device_descriptor dev_descr
- char * ugenpath
- char * phypath

33.127.1 Field Documentation

uint8_t cfgvalue

uint8_t* raw_cfgdescr

struct libusb_device_descriptor dev_descr

char* ugenpath

char* phypath

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.128 sunos_transfer_priv Struct Reference

```
#include <sunos_usb.h>
```

Data Fields

- struct aiocb aiocb
- struct libusb_transfer * transfer

33.128.1 Field Documentation

struct aiocb aiocb

struct libusb_transfer* transfer

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

- lib/ndkbuild/libusb/libusb/os/sunos_usb.h

33.129 tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- USB_DK_DEVICE_ID ID
- ULONG64 Index

33.129.1 Field Documentation

USB_DK_DEVICE_ID ID

ULONG64 Index

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.130 tag_USB_DK_DEVICE_ID Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- WCHAR DeviceID [MAX_DEVICE_ID_LEN]
- WCHAR InstanceID [MAX_DEVICE_ID_LEN]

33.130.1 Field Documentation

WCHAR DeviceID[MAX_DEVICE_ID_LEN]

WCHAR InstanceID[MAX_DEVICE_ID_LEN]

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.131 tag_USB_DK_DEVICE_INFO Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- USB_DK_DEVICE_ID ID
- ULONG64 FilterID
- ULONG64 Port
- ULONG64 Speed
- USB_DEVICE_DESCRIPTOR DeviceDescriptor

33.131.1 Field Documentation

USB_DK_DEVICE_ID ID

ULONG64 FilterID

ULONG64 Port

ULONG64 Speed

USB_DEVICE_DESCRIPTOR DeviceDescriptor

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.132 tag_USB_DK_GEN_TRANSFER_RESULT Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- ULONG64 BytesTransferred
- ULONG64 UsbdStatus

33.132.1 Field Documentation

ULONG64 BytesTransferred

ULONG64 UsbdStatus

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.133 tag_USB_DK_ISO_TRANSFER_RESULT Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- ULONG64 ActualLength
- ULONG64 TransferResult

33.133.1 Field Documentation

ULONG64 ActualLength

ULONG64 TransferResult

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.134 tag_USB_DK_TRANSFER_REQUEST Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- ULONG64 EndpointAddress
- PVOID64 Buffer
- ULONG64 BufferLength
- ULONG64 TransferType
- ULONG64 IsochronousPacketsArraySize
- PVOID64 IsochronousPacketsArray
- USB_DK_TRANSFER_RESULT Result

33.134.1 Field Documentation

ULONG64 EndpointAddress

PVOID64 Buffer

ULONG64 BufferLength

ULONG64 TransferType

ULONG64 IsochronousPacketsArraySize

PVOID64 IsochronousPacketsArray

USB_DK_TRANSFER_RESULT Result

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.135 tag_USB_DK_TRANSFER_RESULT Struct Reference

```
#include <windows_usbdk.h>
```

Data Fields

- USB_DK_GEN_TRANSFER_RESULT GenResult
- PVOID64 IsochronousResultsArray

33.135.1 Field Documentation

USB_DK_GEN_TRANSFER_RESULT GenResult

PVOID64 IsochronousResultsArray

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

- lib/ndkbuild/libusb/libusb/os/windows_usbdk.h

33.136 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

33.136.1 Constructor & Destructor Documentation

∼TestCaseSetup ( )

TestCaseSetup ( )

TestCaseSetup (int *deviceIndex,* int *numChannels*)

33.136.2 Member Function Documentation

void findDevice (void)

void findDevice (AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *find*)

void doSomething ( )

void setCurrentDeviceIndex (int *DeviceIndex*)

void doBulkConfigBlock ( )

Uploads a bulk configuration block.

void doPreSetup ( )

void doSetAutoCalibration (void)

Sets up the :auto: calibration mode.

void doVerifyGroundCalibration (void)

void doVerifyReferenceCalibration (void)

Verify that A/D reference calibration is correct.

void doDemonstrateReadVoltages ( )

DEMONSTRATE SCANNING CHANNELS AND MEASURING VOLTAGES.

void doScanSingleChannel ( )

demonstrate reading a single channel in volts

void doPreReadImmediateVoltages ( )

Performs an immediate read of voltages.

void doCSVReadVoltages ( )

Simple version that just outputs data to csv file.

void doCSVWithGetChannelV ( )

void doCleanupAfterBulk ( )

void doDACDirect (int *channel,* unsigned short *voltage*)

void doDACDirectSetup ( )

void writeBuffer (char ∗ *filename*)

writes the bytes to a file in question.

Will be binary unless the user specifies CSV as an argument

void setMaxCount (int *val*)

void ThrowError (unsigned long *result,* int *linnum*)

Exception handler.

Parameters

result	
linnum	

void doFastITScanSetup ( )

void doFastITScan (int *numgets*)

unsigned short * doGetBuffer ( )

void THROW_IF_ERROR (int *result,* const char * *format,* ...) [static]

int envGetInteger (const char * *env*) [static]

double envGetDouble (const char * *env*) [static]

void doTestSetAutoCalibration (void)

void doGenericVendorWrite (unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* unsigned long * *DataSize,* void * *pData*)

void doBulkAcquire (void)

Demonstrate bulk acquire.

void doBulkAcquire (unsigned int *blck_size,* unsigned int *ovr_sampl,* unsigned int *clk_speed*)

Demonstrate bulk acquire.

Parameters

blck_size	
ovr_sampl	
clk_speed	

void doDisplayBulkResults ( )

void resetCPU (void)

double * getVolts ( )

unsigned short * getCounts ( )

unsigned char * getGainCodes ( )

void setupVoltageParameters (void) [private]

sets up the voltage parameters for runs

unsigned long TEST_ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*) [private]

33.136.3 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

33.137 timer_request Struct Reference

Data Fields

- struct timespec * tp
- HANDLE event

33.137.1 Field Documentation

struct timespec* tp

HANDLE event

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

- lib/ndkbuild/libusb/libusb/os/windows_nt_common.c

33.138 timespec Struct Reference

```
#include <threads_windows.h>
```

Data Fields

- long tv_sec
- long tv_nsec

33.138.1 Field Documentation

long tv_sec

long tv_nsec

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

- lib/ndkbuild/libusb/libusb/os/threads_windows.h

33.139 UCharArray Class Reference

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

Public Member Functions

- UCharArray (int size=0)

33.139.1 Constructor & Destructor Documentation

UCharArray (int *size* = 0) [inline]

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

- deprecated/classlib/USBDeviceBase.hpp

33.140 UKW_CONTROL_HEADER Struct Reference

```
#include <wince_usb.h>
```

Data Fields

- UINT8 bmRequestType
- UINT8 bRequest
- UINT16 wValue
- UINT16 wIndex
- UINT16 wLength

33.140.1 Field Documentation

UINT8 bmRequestType

UINT8 bRequest

UINT16 wValue

UINT16 wIndex

UINT16 wLength

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

- lib/ndkbuild/libusb/libusb/os/wince_usb.h

33.141 UKW_DEVICE_DESCRIPTOR Struct Reference

```
#include <wince_usb.h>
```

Data Fields

- UINT8 bLength
- UINT8 bDescriptorType
- UINT16 bcdUSB
- UINT8 bDeviceClass
- UINT8 bDeviceSubClass
- UINT8 bDeviceProtocol
- UINT8 bMaxPacketSize0
- UINT16 idVendor
- UINT16 idProduct
- UINT16 bcdDevice
- UINT8 iManufacturer
- UINT8 iProduct
- UINT8 iSerialNumber
- UINT8 bNumConfigurations

33.141.1 Field Documentation

UINT8 bLength

UINT8 bDescriptorType

UINT16 bcdUSB

UINT8 bDeviceClass

UINT8 bDeviceSubClass

UINT8 bDeviceProtocol

UINT8 bMaxPacketSize0

UINT16 idVendor

UINT16 idProduct

UINT16 bcdDevice

UINT8 iManufacturer

UINT8 iProduct

UINT8 iSerialNumber

UINT8 bNumConfigurations

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

- lib/ndkbuild/libusb/libusb/os/wince_usb.h

33.142 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

33.142.1 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_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 <i>USB_AI16_Family</i> are automatically created by the USB device manager when they are
* detected on the bus. You should use one of the <i>USBDeviceManager</i> search methods, such as
* <i>USBDeviceManager::getDeviceByProductID( int productID ) const</i>,
* to obtain a reference to a <i>USB_AI16_Family</i> instance. You can then cast the <i>USBDeviceBase</i>
* reference obtained from one of those methods to a <i>USB_AI16_Family</i> and make use of this class' methods, l
* <pre>USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_AI12_32A, USB_AI12_32E );
*if( devices.size() > 0 )
*  USB_AI16_Family &device = *( USB_AI16_Family * ) devices.at( 0 );</pre>
*
```

33.142.2 Constructor & Destructor Documentation

USB_AI16_Family (int *productID*, int *deviceIndex*) `[protected]`

~USB_AI16_Family ( ) `[protected],[virtual]`

33.142.3 Member Function Documentation

void initialize ( ) `[static],[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AnalogInputSubsystem& adc ( ) `[inline]`

Gets a reference to the analog input subsystem of this device.

Returns

A reference to the analog input subsystem.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) `[inline]`

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

33.142.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

33.142.5 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

33.143 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 analogOutputSubsytem
- DigitalIOSubsystem digitalIOSubsystem
- CounterSubsystem counterSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

33.143.1 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 );
```

33.143.2 Constructor & Destructor Documentation

USB_AIO16_Family (int *productID*, int *deviceIndex*)

~USB_AIO16_Family ( ) [virtual]

33.143.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AnalogInputSubsystem& adc ( ) [inline]

Gets a reference to the analog input subsystem of this device.

Returns

A reference to the analog input subsystem.

AnalogOutputSubsystem& dac ( ) [inline]

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) [inline]

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

33.143.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

33.143.5 Field Documentation

IntArray supportedProductIDs [static],[private]

AnalogInputSubsystem analogInputSubsystem [protected]

AnalogOutputSubsystem analogOutputSubsytem [protected]

DigitalIOSubsystem digitalIOSubsystem [protected]

CounterSubsystem counterSubsystem [protected]

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

- deprecated/classlib/USB_AIO16_Family.hpp
- deprecated/classlib/USB_AIO16_Family.cpp

33.144 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

33.144.1 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 );
```

33.144.2 Constructor & Destructor Documentation

USB_AO16_Family (int *productID,* int *deviceIndex*) `[protected]`

~USB_AO16_Family ( ) `[protected],[virtual]`

33.144.3 Member Function Documentation

void initialize ( ) `[static],[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AO16_AnalogOutputSubsystem& dac ( ) [inline]

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

33.144.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

33.144.5 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

33.145 USB_CONFIGURATION_DESCRIPTOR Struct Reference

```
#include <windows_nt_common.h>
```

Data Fields

- UCHAR bLength
- UCHAR bDescriptorType
- USHORT wTotalLength
- UCHAR bNumInterfaces
- UCHAR bConfigurationValue
- UCHAR iConfiguration
- UCHAR bmAttributes
- UCHAR MaxPower

33.145.1 Field Documentation

UCHAR bLength

UCHAR bDescriptorType

USHORT wTotalLength

UCHAR bNumInterfaces

UCHAR bConfigurationValue

UCHAR iConfiguration

UCHAR bmAttributes

UCHAR MaxPower

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

- lib/ndkbuild/libusb/libusb/os/windows_nt_common.h

33.146 USB_CONFIGURATION_DESCRIPTOR_SHORT Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- struct {
    ULONG ConnectionIndex
    struct {
      UCHAR bmRequest
      UCHAR bRequest
      USHORT wValue
      USHORT wIndex
      USHORT wLength
    } SetupPacket
  } req

- USB_CONFIGURATION_DESCRIPTOR data

33.146.1 Field Documentation

ULONG ConnectionIndex

UCHAR bmRequest

UCHAR bRequest

USHORT wValue

USHORT wIndex

USHORT wLength

struct { ... } SetupPacket

struct { ... } req

USB_CONFIGURATION_DESCRIPTOR data

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.147 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

33.147.1 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 );
```

33.147.2 Constructor & Destructor Documentation

USB_CTR_15_Family (int *productID,* int *deviceIndex*) `[protected]`

~USB_CTR_15_Family ( ) `[protected],[virtual]`

33.147.3 Member Function Documentation

void initialize ( ) `[static],[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

CounterSubsystem& ctr ( ) [inline]

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

33.147.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

33.147.5 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

33.148 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

33.148.1 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 );
```

33.148.2 Constructor & Destructor Documentation

USB_DA12_8A_Family (int *productID*, int *deviceIndex*) [protected]

~USB_DA12_8A_Family ( ) [protected],[virtual]

33.148.3 Member Function Documentation

void initialize ( ) [static],[private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DA12_AnalogOutputSubsystem& dac ( ) `[inline]`

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

33.148.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

33.148.5 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

33.149 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

33.149.1 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 );
```

33.149.2 Constructor & Destructor Documentation

USB_DA12_8E_Family (int *productID*, int *deviceIndex*) `[protected]`

~USB_DA12_8E_Family ( ) `[protected],[virtual]`

33.149.3 Member Function Documentation

void initialize ( ) `[static],[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DA12_AnalogOutputSubsystem& dac ( ) `[inline]`

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

33.149.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

33.149.5 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

33.150 usb_descriptor_header Struct Reference

```
#include <libusbi.h>
```

Data Fields

- uint8_t bLength
- uint8_t bDescriptorType

33.150.1 Field Documentation

uint8_t bLength

uint8_t bDescriptorType

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.151 USB_DESCRIPTOR_REQUEST Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ConnectionIndex
- struct {
  UCHAR bmRequest
  UCHAR bRequest
  USHORT wValue
  USHORT wIndex
  USHORT wLength
  } SetupPacket

33.151.1 Field Documentation

ULONG ConnectionIndex

UCHAR bmRequest

UCHAR bRequest

USHORT wValue

USHORT wIndex

USHORT wLength

struct { ... } SetupPacket

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.152 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

33.152.1 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 );
```

33.152.2 Constructor & Destructor Documentation

USB_DIO_16_Family (int *productID,* int *deviceIndex*) [protected]

~USB_DIO_16_Family ( ) [protected],[virtual]

33.152.3 Member Function Documentation

void initialize ( ) [static],[private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

DIOStreamSubsystem& diostream ( ) [inline]

Gets a reference to the digital I/O streaming subsystem of this device.

Returns

A reference to the digital I/O streaming subsystem.

33.152.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

33.152.5 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

33.153 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

33.153.1 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 );
```

33.153.2 Constructor & Destructor Documentation

USB_DIO_32_Family (int *productID,* int *deviceIndex*) [protected]

~**USB_DIO_32_Family** ( ) [protected],[virtual]

33.153.3 Member Function Documentation

void initialize ( ) [static],[private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) [inline]

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

33.153.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

33.153.5 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

33.154 USB_DIO_Family Class Reference

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

```
#include <USB_DIO_Family.hpp>
```

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DIO_Family (int productID, int deviceIndex)
- virtual ~USB_DIO_Family ()

Protected Attributes

- DigitalIOSubsystem digitalIOSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

33.154.1 Detailed Description

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

Instances of class *USB_DIO_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DIO_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DIO_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DIO_48, USB_DIO_96 );
if( devices.size() > 0 )
  USB_DIO_Family &device = *( USB_DIO_Family * ) devices.at( 0 );
```

33.154.2 Constructor & Destructor Documentation

USB_DIO_Family (int *productID*, int *deviceIndex*) `[protected]`

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

33.154.3 Member Function Documentation

void initialize ( ) `[static],[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

33.154.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

33.154.5 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

33.155 USB_ENDPOINT_DESCRIPTOR Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- UCHAR bLength
- UCHAR bDescriptorType
- UCHAR bEndpointAddress
- UCHAR bmAttributes
- USHORT wMaxPacketSize
- UCHAR bInterval

33.155.1 Field Documentation

UCHAR bLength

UCHAR bDescriptorType

UCHAR bEndpointAddress

UCHAR bmAttributes

USHORT wMaxPacketSize

UCHAR bInterval

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.156 USB_HUB_CAP_FLAGS Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG HubIsHighSpeedCapable:1
- ULONG HubIsHighSpeed:1
- ULONG HubIsMultiTtCapable:1
- ULONG HubIsMultiTt:1
- ULONG HubIsRoot:1
- ULONG HubIsArmedWakeOnConnect:1
- ULONG ReservedMBZ:26

33.156.1 Field Documentation

ULONG HubIsHighSpeedCapable

ULONG HubIsHighSpeed

ULONG HubIsMultiTtCapable

ULONG HubIsMultiTt

ULONG HubIsRoot

ULONG HubIsArmedWakeOnConnect

ULONG ReservedMBZ

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.157 USB_HUB_CAPABILITIES Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG HubIs2xCapable:1

33.157.1 Field Documentation

ULONG HubIs2xCapable

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.158 USB_HUB_CAPABILITIES_EX Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USB_HUB_CAP_FLAGS CapabilityFlags

33.158.1 Field Documentation

USB_HUB_CAP_FLAGS CapabilityFlags

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.159 USB_HUB_DESCRIPTOR Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- UCHAR bDescriptorLength
- UCHAR bDescriptorType
- UCHAR bNumberOfPorts
- USHORT wHubCharacteristics
- UCHAR bPowerOnToPowerGood
- UCHAR bHubControlCurrent
- UCHAR bRemoveAndPowerMask [64]

33.159.1 Field Documentation

UCHAR bDescriptorLength

UCHAR bDescriptorType

UCHAR bNumberOfPorts

USHORT wHubCharacteristics

UCHAR bPowerOnToPowerGood

UCHAR bHubControlCurrent

UCHAR bRemoveAndPowerMask[64]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.160 USB_HUB_INFORMATION Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USB_HUB_DESCRIPTOR HubDescriptor
- BOOLEAN HubIsBusPowered

33.160.1 Field Documentation

USB_HUB_DESCRIPTOR HubDescriptor

BOOLEAN HubIsBusPowered

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.161 USB_HUB_NAME_FIXED Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- union {
  USB_ROOT_HUB_NAME_FIXED root
  USB_NODE_CONNECTION_NAME_FIXED node
  } u

33.161.1 Field Documentation

USB_ROOT_HUB_NAME_FIXED root

USB_NODE_CONNECTION_NAME_FIXED node

union { ... } u

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.162 USB_INTERFACE_DESCRIPTOR Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- UCHAR bLength
- UCHAR bDescriptorType
- UCHAR bInterfaceNumber
- UCHAR bAlternateSetting
- UCHAR bNumEndpoints
- UCHAR bInterfaceClass
- UCHAR bInterfaceSubClass
- UCHAR bInterfaceProtocol
- UCHAR iInterface

33.162.1 Field Documentation

UCHAR bLength

UCHAR bDescriptorType

UCHAR bInterfaceNumber

UCHAR bAlternateSetting

UCHAR bNumEndpoints

UCHAR bInterfaceClass

UCHAR bInterfaceSubClass

UCHAR bInterfaceProtocol

UCHAR iInterface

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.163 USB_MI_PARENT_INFORMATION Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG NumberOfInterfaces

33.163.1 Field Documentation

ULONG NumberOfInterfaces

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.164 USB_NODE_CONNECTION_INFORMATION_EX Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ConnectionIndex
- USB_DEVICE_DESCRIPTOR DeviceDescriptor
- UCHAR CurrentConfigurationValue
- UCHAR Speed
- BOOLEAN DeviceIsHub
- USHORT DeviceAddress
- ULONG NumberOfOpenPipes
- USB_CONNECTION_STATUS ConnectionStatus

33.164.1 Field Documentation

ULONG ConnectionIndex

USB_DEVICE_DESCRIPTOR DeviceDescriptor

UCHAR CurrentConfigurationValue

UCHAR Speed

BOOLEAN DeviceIsHub

USHORT DeviceAddress

ULONG NumberOfOpenPipes

USB_CONNECTION_STATUS ConnectionStatus

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.165 USB_NODE_CONNECTION_NAME Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ConnectionIndex
- ULONG ActualLength
- WCHAR NodeName [1]

33.165.1 Field Documentation

ULONG ConnectionIndex

ULONG ActualLength

WCHAR NodeName[1]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.166 USB_NODE_CONNECTION_NAME_FIXED Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ConnectionIndex
- ULONG ActualLength
- WCHAR NodeName [MAX_PATH_LENGTH]

33.166.1 Field Documentation

ULONG ConnectionIndex

ULONG ActualLength

WCHAR NodeName[MAX_PATH_LENGTH]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.167 USB_NODE_INFORMATION Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USB_HUB_NODE NodeType
- union {
  USB_HUB_INFORMATION HubInformation
  USB_MI_PARENT_INFORMATION MiParentInformation
  } u

33.167.1 Field Documentation

USB_HUB_NODE NodeType

USB_HUB_INFORMATION HubInformation

USB_MI_PARENT_INFORMATION MiParentInformation

union { ... } u

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.168 USB_PIPE_INFO Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USB_ENDPOINT_DESCRIPTOR EndpointDescriptor
- ULONG ScheduleOffset

33.168.1 Field Documentation

USB_ENDPOINT_DESCRIPTOR EndpointDescriptor

ULONG ScheduleOffset

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.169 usb_raw_command Union Reference

```
#include <haiku_usb_raw.h>
```

Data Fields

- struct {
  status_t status
  } version

- struct {
  status_t status
  usb_device_descriptor * descriptor
  } device

- struct {
  status_t status
  usb_configuration_descriptor * descriptor
  uint32 config_index
  } config

- struct {
  status_t status
  uint32 alternate_info
  uint32 config_index
  uint32 interface_index
  } alternate

- struct {
  status_t status
  usb_interface_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  } interface

- struct {
  status_t status
  usb_interface_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  uint32 alternate_index
  } interface_etc

- struct {
  status_t status
  usb_endpoint_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  uint32 endpoint_index
  } endpoint

- struct {
  status_t status
  usb_endpoint_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  uint32 alternate_index
  uint32 endpoint_index
  } endpoint_etc

- struct {
  status_t status
  usb_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  uint32 generic_index
  size_t length
  } generic

- struct {
  status_t status
  usb_descriptor * descriptor
  uint32 config_index
  uint32 interface_index
  uint32 alternate_index
  uint32 generic_index
  size_t length
  } generic_etc

- struct {
  status_t status
  usb_string_descriptor * descriptor
  uint32 string_index
  size_t length
  } string

- struct {
  status_t status
  uint8 type
  uint8 index
  uint16 language_id
  void * data
  size_t length
  } descriptor

- struct {
  status_t status
  uint8 request_type
  uint8 request
  uint16 value
  uint16 index
  uint16 length
  void * data
  } control

- struct {
  status_t status
  uint32 interface
  uint32 endpoint
  void * data
  size_t length
  } transfer

- struct {
  status_t status
  uint32 interface
  uint32 endpoint
  void * data
  size_t length
  usb_iso_packet_descriptor * packet_descriptors
  uint32 packet_count
  } isochronous

33.169.1 Field Documentation

status_t status

struct { ... } version

usb_device_descriptor* descriptor

struct { ... } device

usb_configuration_descriptor* descriptor

uint32 config_index

struct { ... } config

uint32 alternate_info

uint32 interface_index

struct { ... } alternate

usb_interface_descriptor* descriptor

struct { ... } interface

uint32 alternate_index

struct { ... } interface_etc

usb_endpoint_descriptor* descriptor

uint32 endpoint_index

struct { ... } endpoint

struct { ... } endpoint_etc

usb_descriptor* descriptor

uint32 generic_index

size_t length

struct { ... } generic

struct { ... } generic_etc

usb_string_descriptor* descriptor

uint32 string_index

struct { ... } string

uint8 type

uint8 index

uint16 language_id

void* data

struct { ... } descriptor

uint8 request_type

uint8 request

uint16 value

uint16 index

uint16 length

struct { ... } control

uint32 interface

uint32 endpoint

struct { ... } transfer

usb_iso_packet_descriptor* packet_descriptors

uint32 packet_count

struct { ... } isochronous

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

- lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.h

33.170 USB_ROOT_HUB_NAME Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ActualLength
- WCHAR RootHubName [1]

33.170.1 Field Documentation

ULONG ActualLength

WCHAR RootHubName[1]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.171 USB_ROOT_HUB_NAME_FIXED Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- ULONG ActualLength
- WCHAR RootHubName [MAX_PATH_LENGTH]

33.171.1 Field Documentation

ULONG ActualLength

WCHAR RootHubName[MAX_PATH_LENGTH]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.172 USBDevice Struct Reference

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

Public Member Functions

- USBDevice (const char ∗)
- virtual ∼USBDevice ()
- const char ∗ Location () const
- uint8 CountConfigurations () const
- const usb_device_descriptor ∗ Descriptor () const
- const usb_configuration_descriptor ∗ ConfigurationDescriptor (uint32) const
- const usb_configuration_descriptor ∗ ActiveConfiguration () const
- uint8 EndpointToIndex (uint8) const
- uint8 EndpointToInterface (uint8) const
- int ClaimInterface (int)
- int ReleaseInterface (int)
- int CheckInterfacesFree (int)
- int SetActiveConfiguration (int)
- int ActiveConfigurationIndex () const
- bool InitCheck ()

Data Fields

- int(∗ usb_control_transfer)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int(∗ usb_bulk_transfer)(USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int(∗ usb_request)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int(∗ usb_reset_device)(USBDevice ∗usbdev)
- int(∗ usb_put_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int(∗ usb_get_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- uint8_t timeout
- libusb_device ∗ device
- libusb_device_handle ∗ deviceHandle
- struct libusb_device_descriptor deviceDesc
- AIOUSB_BOOL debug
- int usblp_attached
- int iface
- int verbose
- int conf
- int origconf
- int altset

Private Member Functions

- int Initialise ()

Private Attributes

- unsigned int fClaimedInterfaces
- usb_device_descriptor fDeviceDescriptor
- unsigned char ∗∗ fConfigurationDescriptors
- int fActiveConfiguration
- char ∗ fPath
- map< uint8, uint8 > fConfigToIndex
- map< uint8, uint8 > ∗ fEndpointToIndex
- map< uint8, uint8 > ∗ fEndpointToInterface
- bool fInitCheck

33.172.1 Constructor & Destructor Documentation

USBDevice (**const** char ∗ *path*)

~**USBDevice** ( ) [virtual]

33.172.2 Member Function Documentation

const char ∗ Location ( ) const

uint8 CountConfigurations ( ) const

const usb_device_descriptor ∗ Descriptor ( ) const

const usb_configuration_descriptor ∗ ConfigurationDescriptor (uint32 *index*) const

const usb_configuration_descriptor ∗ ActiveConfiguration ( ) const

uint8 EndpointToIndex (uint8 *address*) const

uint8 EndpointToInterface (uint8 *address*) const

int ClaimInterface (int *interface*)

int ReleaseInterface (int *interface*)

int CheckInterfacesFree (int *config*)

int SetActiveConfiguration (int *config_index*)

int ActiveConfigurationIndex ( ) const

bool InitCheck ( )

int Initialise ( ) [private]

33.172.3 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

unsigned int fClaimedInterfaces [private]

usb_device_descriptor fDeviceDescriptor [private]

unsigned char fConfigurationDescriptors** [private]

int fActiveConfiguration [private]

char* fPath [private]

map<uint8,uint8> fConfigToIndex [private]

map<uint8,uint8>* fEndpointToIndex [private]

map<uint8,uint8>* fEndpointToInterface [private]

bool fInitCheck [private]

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

- lib/ndkbuild/jni/USBDevice.h
- lib/ndkbuild/libusb/libusb/os/haiku_usb.h
- lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp

33.173 USBDeviceArray Class Reference

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

Public Member Functions

- USBDeviceArray (int size=0)

33.173.1 Constructor & Destructor Documentation

USBDeviceArray (int *size* = 0) [inline]

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

- deprecated/classlib/USBDeviceBase.hpp

33.174 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

33.174.1 Detailed Description

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

33.174.2 Constructor & Destructor Documentation

USBDeviceBase (int *productID,* int *deviceIndex*) `[protected]`

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

33.174.3 Member Function Documentation

USBDeviceBase & clearFIFO (FIFO_Method *method*) `[protected]`

double getMiscClock ( ) `[inline],[protected]`

USBDeviceBase & setMiscClock (double *clockHz*) [protected]

int getStreamingBlockSize ( ) [protected]

USBDeviceBase & setStreamingBlockSize (int *blockSize*) [protected]

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

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented in USB_AI16_Family, USB_DIO_Family, USB_AO16_Family, USB_DA12_8A_Family, USB_DIO_16_-Family, USB_DIO_32_Family, USB_AIO16_Family, USB_DA12_8E_Family, and USB_CTR_15_Family.

int getDeviceIndex ( ) const [inline]

Gets the device's index on the USB bus.

The device index isn't used within this Java class library, but is used in the underlying AIOUSB library. The device index is somewhat useful within this Java class library to differentiate between multiple devices of the same type.

Returns

The index of the device on the USB bus.

int getProductID ( ) const [inline]

Gets the device's product ID.

Returns

The device product ID.

const std::string& getName ( ) const [inline]

Gets the device's name.

Returns

The device name.

__uint64_t getSerialNumber ( ) const [inline]

Gets the device's serial number.

Returns

The device serial number (a 64-bit integer).

Exceptions

OperationFailedException	

int getCommTimeout ( ) const

Gets the current timeout setting for USB communications.

Returns

Current timeout setting (in milliseconds).

See Also

setCommTimeout(int timeout)

USBDeviceBase & setCommTimeout (int *timeout*)

Sets the timeout for USB communications.

Parameters

timeout	the new timeout setting (in milliseconds; default is 5,000).

Returns

This device, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

USBDeviceBase & reset ( )

Perform a USB port reset to reinitialize the device.

Returns

This device, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceBase & customEEPROMWrite (int *address*, const UCharArray & *data*)

Writes data to the custom programming area of the device EEPROM.

Beware that writing to the EEPROM is particularly slow. Writing the entire EEPROM may take several seconds. Before initiating a lengthy EEPROM write procedure, it is recommended that the communication timeout be increased to at least five seconds, if not ten *(see setCommTimeout(int timeout))*. Otherwise, a timeout error will occur before the write procedure finishes. Once the write procedure is finished, you can restore the timeout to a more reasonable value. If you are writing a smaller amount of data to the EEPROM, you may reduce the timeout proportionately.

Parameters

address	starting address from 0x000 to 0x1FF within the EEPROM.
data	an array of bytes containing the data to write to the EEPROM, beginning at the starting address. The starting address plus the data size may not exceed the maximum address of 0x1FF.

Returns

This device, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

UCharArray customEEPROMRead (int *address*, int *numBytes*)

Reads data from the custom programming area of the device EEPROM.

Parameters

address	starting address from 0x000 to 0x1FF within the EEPROM.
numBytes	the number of bytes to read from the EEPROM, beginning at the starting address. The starting address plus the number of bytes to read may not exceed the maximum address of 0x1FF.

Returns

An array of bytes containing the data read from the EEPROM. The length of the array will be equal to *numBytes*.

Exceptions

IllegalArgumentException	
OperationFailedException	

33.174.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

friend class DIOStreamSubsystem [friend]

friend class AnalogInputSubsystem [friend]

33.174.5 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

33.175 USBDeviceHandle Class Reference

```
#include <haiku_usb.h>
```

Public Member Functions

- USBDeviceHandle (USBDevice *dev)
- virtual ~USBDeviceHandle ()
- int ClaimInterface (int)
- int ReleaseInterface (int)
- int SetConfiguration (int)
- int SetAltSetting (int, int)
- status_t SubmitTransfer (struct usbi_transfer *)
- status_t CancelTransfer (USBTransfer *)
- bool InitCheck ()

Private Member Functions

- void TransfersWorker ()

Static Private Member Functions

- static status_t TransfersThread (void *)

Private Attributes

- int fRawFD
- USBDevice * fUSBDevice
- unsigned int fClaimedInterfaces
- BList fTransfers
- BLocker fTransfersLock
- sem_id fTransfersSem
- thread_id fTransfersThread
- bool fInitCheck

33.175.1 Constructor & Destructor Documentation

USBDeviceHandle (USBDevice * *dev*)

~USBDeviceHandle ( ) [virtual]

33.175.2 Member Function Documentation

int ClaimInterface (int *inumber*)

int ReleaseInterface (int *inumber*)

int SetConfiguration (int *config*)

int SetAltSetting (int *inumber,* int *alt*)

status_t SubmitTransfer (struct usbi_transfer * *itransfer*)

status_t CancelTransfer (USBTransfer * *transfer*)

bool InitCheck ( )

status_t TransfersThread (void * *self*) [static], [private]

void TransfersWorker ( ) [private]

33.175.3 Field Documentation

int fRawFD [private]

USBDevice* fUSBDevice [private]

unsigned int fClaimedInterfaces [private]

BList fTransfers [private]

BLocker fTransfersLock [private]

sem_id fTransfersSem [private]

thread_id fTransfersThread [private]

bool fInitCheck [private]

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

- lib/ndkbuild/libusb/libusb/os/haiku_usb.h
- lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp

33.176 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"

33.176.1 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.

33.176.2 Constructor & Destructor Documentation

USBDeviceManager ( )

~USBDeviceManager ( ) `[virtual]`

33.176.3 Member Function Documentation

void emptyDeviceList ( ) `[protected]`

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

Prints the properties of this device manager and all of the devices found on the bus to the specified print stream.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

USBDeviceManager & printDevices ( )

Prints the properties of this device manager and all of the devices found on the bus to the standard output device.

Mainly useful for diagnostic purposes.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

std::string getAIOUSBVersion ( ) const `[inline]`

Gets the version number of the underlying AIOUSB module.

Returns

The AIOUSB module version number as a string with the form, "1.78".

std::string getAIOUSBVersionDate ( ) const `[inline]`

Gets the version date of the underlying AIOUSB module.

Returns

The AIOUSB module version date as a string with the form, "15 November 2009".

std::string productIDToName (int *productID*) [static]

Gets the product name for a product ID.

This name is only "approximate," as an actual device reports its own name. Generally the names reported by the device are the same as those obtained from this method, but that is not guaranteed. This method provides a name that constitutes a user-friendly alternative to a product ID number. The complement of this method is *productNameToID(const std::string &productName)*. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productID	the product ID to translate to a product name.

Returns

A string containing the product name, or "UNKNOWN" if the product ID was not found.

Exceptions

IllegalArgumentException	

StringArray productIDToName (const IntArray & *productID*) [static]

Gets the product names for an array of product IDs.

Functionally identical to *productIDToName(int productID)* except that it operates on an array of product IDs rather than an individual product ID. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productID	an array of product IDs to translate to product names.

Returns

An array of strings containing the product names, or "UNKNOWN" for any product ID that was not found. The product names are returned in the same order as the product IDs passed in *productID[]*.

Exceptions

IllegalArgumentException	

int productNameToID (const std::string & *productName*) [static]

Gets the product ID for a product name.

This method is the complement of *productIDToName(int productID)* and one should read the notes for that method. It is not guaranteed that *productNameToID()* will successfully ascertain the product ID for a name obtained from a device, although it usually will. *ProductNameToID()* will always successfully ascertain the product ID for a name obtained from *productIDToName()*. If one has access to a device and its name, then they should obtain the product ID from the device itself rather than from this method. This method is mainly for easily converting between product names and IDs, primarily to serve the needs of user interfaces. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productName	the product name to translate to a product ID.

Returns

The product ID for the specified product name, or 0 (zero) if the name was not found.

Exceptions

IllegalArgumentException	

IntArray productNameToID (const StringArray & *productName*) [static]

Gets the product IDs for an array of product names.

Functionally identical to *productNameToID(const std::string &productName)* except that it operates on an array of product names rather than an individual product name. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productName	an array of product names to translate to product IDs.

Returns

An array of integers containing the product IDs, or 0 (zero) for any product name that was not found. The product IDs are returned in the same order as the product names passed in *productName[]*.

Exceptions

IllegalArgumentException	

void listDevices ( ) const [inline]

Prints the properties of all the devices found on the bus to the standard output device.

This function is similar to *printDevices()* but is implemented by the underlying AIOUSB module and produces different output than *printDevices()*. Mainly useful for diagnostic purposes.

static std::string getResultCodeAsString (int *result*) [inline],[static]

Gets the string representation of an AIOUSB result code, useful mainly for debugging purposes.

This method is also used to convert an AIOUSB result code to a string when an OperationFailedException is thrown in response to an AIOUSB failure. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for result code lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

result	an AIOUSB result code.

Returns

The string representation of *result*.

bool isOpen ( ) const [inline]

Tells if the USB device manager has been "opened" for use *(see open())*.

Returns

True indicates that the device manager is open and ready to be used; *false* indicates that it is not open.

USBDeviceManager & open ( )

"Opens" the USB device manager for use.

Before the USB device manager may be used, *open()* must be called. *Open()* initializes the underlying AIOUSB module and scans the bus for devices, building a list of the devices found. When finished using the USB device manager, *close()* must be called. It is possible to call *close()* and then call *open()* again, which effectively reinitializes everything.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceManager & close ( )

"Closes" the USB device manager for use.

When finished using the USB device manager, and assuming *open()* was properly called, *close()* must be called. *Close()* terminates use of the underlying AIOUSB module and discards the list of devices found. *You must terminate use of all USB devices before calling close()!* You can call *open()* again to reinitialize things and reestablish connections to USB devices.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceManager & scanForDevices ( )

Re-scans the bus for devices.

ScanForDevices() is called automatically by *open()*. *You must terminate use of all USB devices before calling scanForDevices()!* After calling *scanForDevices()* you can reestablish connections to USB devices.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceArray getDeviceByProductID (int *productID*) const

Gets a list of all the devices found on the bus matching the specified product ID.

Only devices exactly matching the specified product ID will be returned. You can search for devices by product name using productNameToID(const std::string &productName), like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( deviceManager.productNameToID( "USB-CTR-
```

Parameters

productID	the product ID to search for.

Returns

An array of all the devices found. If no devices were found matching the specified product ID, the array will be empty (i.e. contain zero items).

USBDeviceArray getDeviceByProductID (int *minProductID,* int *maxProductID*) const

Gets a list of all the devices found on the bus matching the specified product ID range.

Any device with a product ID greater than or equal to *minProductID* and less than or equal to *maxProductID* will be returned. You can obtain the entire list of devices detected by passing a value of 0 for *minProductID* and a value of 0xffff for *maxProductID*. Then you can search the list obtained using your own search criteria.

Parameters

minProductID	the minimum product ID to search for.
maxProductID	the maximum product ID to search for.

Returns

An array of all the devices found. If no devices were found matching the specified product ID range, the array will be empty (i.e. contain zero items).

Exceptions

IllegalArgumentException	

USBDeviceArray getDeviceByProductID (const IntArray & *productIDs*) const

Gets a list of all the devices found on the bus matching the specified set of product IDs.

Any device with a product ID equal to one of the products listed in *productIDs[]* will be returned.

Parameters

productIDs	an array containing one or more product IDs to search for.

Returns

An array of all the devices found. If no devices were found matching the specified set of product IDs, the array will be empty (i.e. contain zero items).

Exceptions

IllegalArgumentException	

USBDeviceArray getDeviceBySerialNumber (__uint64_t *serialNumber*) const

Gets a list of all the devices found on the bus matching the specified serial number.

Only devices exactly matching the specified serial number will be returned. In theory, there ought to be only one device matching a given serial number, but this method returns a vector in order to be consistent with the other search methods, and in unlikely event that multiple devices do share the same serial number.

Parameters

serialNumber	the serial number to search for.

Returns

An array of all the devices found. If no devices were found matching the specified serial number, the array will be empty (i.e. contain zero items).

33.176.4 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

33.177 usbfs_bulktransfer Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int ep
- unsigned int len
- unsigned int timeout
- void * data

33.177.1 Field Documentation

unsigned int ep

unsigned int len

unsigned int timeout

void* data

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.178 usbfs_connectinfo Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int devnum
- unsigned char slow

33.178.1 Field Documentation

unsigned int devnum

unsigned char slow

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.179 usbfs_ctrltransfer Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- uint8_t bmRequestType
- uint8_t bRequest
- uint16_t wValue
- uint16_t wIndex
- uint16_t wLength
- uint32_t timeout
- void * data

33.179.1 Field Documentation

uint8_t bmRequestType

uint8_t bRequest

uint16_t wValue

uint16_t wIndex

uint16_t wLength

uint32_t timeout

void* data

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.180 usbfs_disconnect_claim Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int interface
- unsigned int flags
- char driver [USBFS_MAXDRIVERNAME+1]

33.180.1 Field Documentation

unsigned int interface

unsigned int flags

char driver[USBFS_MAXDRIVERNAME+1]

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.181 usbfs_getdriver Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int interface
- char driver [USBFS_MAXDRIVERNAME+1]

33.181.1 Field Documentation

unsigned int interface

char driver[USBFS_MAXDRIVERNAME+1]

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.182 usbfs_hub_portinfo Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned char numports
- unsigned char port [127]

33.182.1 Field Documentation

unsigned char numports

unsigned char port[127]

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.183 usbfs_ioctl Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- int ifno
- int ioctl_code
- void * data

33.183.1 Field Documentation

int ifno

int ioctl_code

void* data

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.184 usbfs_iso_packet_desc Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int length
- unsigned int actual_length
- unsigned int status

33.184.1 Field Documentation

unsigned int length

unsigned int actual_length

unsigned int status

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.185 usbfs_setinterface Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int interface
- unsigned int altsetting

33.185.1 Field Documentation

unsigned int interface

unsigned int altsetting

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.186 usbfs_streams Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned int num_streams
- unsigned int num_eps
- unsigned char eps [0]

33.186.1 Field Documentation

unsigned int num_streams

unsigned int num_eps

unsigned char eps[0]

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.187 usbfs_urb Struct Reference

```
#include <linux_usbfs.h>
```

Data Fields

- unsigned char type
- unsigned char endpoint
- int status
- unsigned int flags
- void * buffer
- int buffer_length
- int actual_length
- int start_frame
- union {
  int number_of_packets
  unsigned int stream_id
  };
- int error_count
- unsigned int signr
- void * usercontext
- struct usbfs_iso_packet_desc iso_frame_desc [0]

33.187.1 Field Documentation

unsigned char type

unsigned char endpoint

int status

unsigned int flags

void* buffer

int buffer_length

int actual_length

int start_frame

int number_of_packets

unsigned int stream_id

union { ... }

int error_count

unsigned int signr

void* usercontext

struct usbfs_iso_packet_desc iso_frame_desc[0]

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

- lib/ndkbuild/libusb/libusb/os/linux_usbfs.h

33.188 usbi_cond Struct Reference

```
#include <threads_windows.h>
```

Data Fields

- struct list_head waiters
- struct list_head not_waiting

33.188.1 Field Documentation

struct list_head waiters

struct list_head not_waiting

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

- lib/ndkbuild/libusb/libusb/os/threads_windows.h

33.189 usbi_cond_perthread Struct Reference

Data Fields

- struct list_head list
- DWORD tid
- HANDLE event

33.189.1 Field Documentation

struct list_head list

DWORD tid

HANDLE event

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

- lib/ndkbuild/libusb/libusb/os/threads_windows.c

33.190 usbi_os_backend Struct Reference

```
#include <libusbi.h>
```

Data Fields

- const char * name
- uint32_t caps
- int(* init)(struct libusb_context *ctx)
- void(* exit)(void)
- int(* get_device_list)(struct libusb_context *ctx, struct discovered_devs **discdevs)
- void(* hotplug_poll)(void)
- int(* open)(struct libusb_device_handle *dev_handle)
- void(* close)(struct libusb_device_handle *dev_handle)
- int(* get_device_descriptor)(struct libusb_device *device, unsigned char *buffer, int *host_endian)
- int(* get_active_config_descriptor)(struct libusb_device *device, unsigned char *buffer, size_t len, int *host_endian)
- int(* get_config_descriptor)(struct libusb_device *device, uint8_t config_index, unsigned char *buffer, size_t len, int *host_endian)
- int(* get_config_descriptor_by_value)(struct libusb_device *device, uint8_t bConfigurationValue, unsigned char **buffer, int *host_endian)
- int(* get_configuration)(struct libusb_device_handle *dev_handle, int *config)
- int(* set_configuration)(struct libusb_device_handle *dev_handle, int config)
- int(* claim_interface)(struct libusb_device_handle *dev_handle, int interface_number)
- int(* release_interface)(struct libusb_device_handle *dev_handle, int interface_number)
- int(* set_interface_altsetting)(struct libusb_device_handle *dev_handle, int interface_number, int altsetting)
- int(* clear_halt)(struct libusb_device_handle *dev_handle, unsigned char endpoint)
- int(* reset_device)(struct libusb_device_handle *dev_handle)
- int(* alloc_streams)(struct libusb_device_handle *dev_handle, uint32_t num_streams, unsigned char *endpoints, int num_endpoints)
- int(* free_streams)(struct libusb_device_handle *dev_handle, unsigned char *endpoints, int num_endpoints)
- unsigned char *(* dev_mem_alloc)(struct libusb_device_handle *handle, size_t len)
- int(* dev_mem_free)(struct libusb_device_handle *handle, unsigned char *buffer, size_t len)
- int(* kernel_driver_active)(struct libusb_device_handle *dev_handle, int interface_number)
- int(* detach_kernel_driver)(struct libusb_device_handle *dev_handle, int interface_number)
- int(* attach_kernel_driver)(struct libusb_device_handle *dev_handle, int interface_number)
- void(* destroy_device)(struct libusb_device *dev)
- int(* submit_transfer)(struct usbi_transfer *itransfer)
- int(* cancel_transfer)(struct usbi_transfer *itransfer)
- void(* clear_transfer_priv)(struct usbi_transfer *itransfer)
- int(* handle_events)(struct libusb_context *ctx, struct pollfd *fds, POLL_NFDS_TYPE nfds, int num_ready)
- int(* handle_transfer_completion)(struct usbi_transfer *itransfer)
- int(* clock_gettime)(int clkid, struct timespec *tp)
- size_t device_priv_size
- size_t device_handle_priv_size
- size_t transfer_priv_size

33.190.1 Field Documentation

const char* name

uint32_t caps

int(* init)(struct **libusb_context** *ctx)

void(* exit)(**void**)

int(* get_device_list)(struct **libusb_context** *ctx, struct **discovered_devs** **discdevs)

void(* hotplug_poll)(**void**)

int(* open)(struct **libusb_device_handle** *dev_handle)

void(* close)(struct **libusb_device_handle** *dev_handle)

int(* get_device_descriptor)(struct **libusb_device** *device, unsigned char *buffer, int *host_endian)

int(* get_active_config_descriptor)(struct **libusb_device** *device, unsigned char *buffer, size_t len, int *host_endian)

int(* get_config_descriptor)(struct **libusb_device** *device, **uint8_t** config_index, unsigned char *buffer, size_t len, int *host_endian)

int(* get_config_descriptor_by_value)(struct **libusb_device** *device, **uint8_t** bConfigurationValue, unsigned char **buffer, int *host_endian)

int(* get_configuration)(struct **libusb_device_handle** *dev_handle, int *config)

int(* set_configuration)(struct **libusb_device_handle** *dev_handle, int config)

int(* claim_interface)(struct **libusb_device_handle** *dev_handle, int interface_number)

int(* release_interface)(struct **libusb_device_handle** *dev_handle, int interface_number)

int(* set_interface_altsetting)(struct **libusb_device_handle** *dev_handle, int interface_number, int altsetting)

int(* clear_halt)(struct **libusb_device_handle** *dev_handle, unsigned char **endpoint**)

int(* reset_device)(struct **libusb_device_handle** *dev_handle)

int(* alloc_streams)(struct **libusb_device_handle** *dev_handle, **uint32_t** num_streams, unsigned char *endpoints, int num_endpoints)

int(* free_streams)(struct **libusb_device_handle** *dev_handle, unsigned char *endpoints, int num_endpoints)

unsigned char*(* dev_mem_alloc)(struct **libusb_device_handle** ***handle**, size_t len)

int(* dev_mem_free)(struct **libusb_device_handle** ***handle**, unsigned char *buffer, size_t len)

int(* kernel_driver_active)(struct **libusb_device_handle** *dev_handle, int interface_number)

int(* detach_kernel_driver)(struct **libusb_device_handle** *dev_handle, int interface_number)

int(* attach_kernel_driver)(struct **libusb_device_handle** *dev_handle, int interface_number)

void(* destroy_device)(struct **libusb_device** *dev)

int(* submit_transfer)(struct **usbi_transfer** *itransfer)

int(* cancel_transfer)(struct **usbi_transfer** *itransfer)

void(* clear_transfer_priv)(struct **usbi_transfer** *itransfer)

int(* handle_events)(struct **libusb_context** *ctx, struct **pollfd** *fds, **POLL_NFDS_TYPE** nfds, int num_ready)

int(* handle_transfer_completion)(struct **usbi_transfer** *itransfer)

int(* clock_gettime)(int clkid, struct timespec *tp)

size_t device_priv_size

size_t device_handle_priv_size

size_t transfer_priv_size

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.191 usbi_pollfd Struct Reference

```
#include <libusbi.h>
```

Data Fields

- struct libusb_pollfd pollfd
- struct list_head list

33.191.1 Field Documentation

struct libusb_pollfd pollfd

struct list_head list

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.192 usbi_transfer Struct Reference

```
#include <libusbi.h>
```

Data Fields

- int num_iso_packets
- struct list_head list
- struct list_head completed_list
- struct timeval timeout
- int transferred
- uint32_t stream_id
- uint8_t state_flags
- uint8_t timeout_flags
- usbi_mutex_t lock

33.192.1 Field Documentation

int num_iso_packets

struct list_head list

struct list_head completed_list

struct timeval timeout

int transferred

uint32_t stream_id

uint8_t state_flags

uint8_t timeout_flags

usbi_mutex_t lock

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

- lib/ndkbuild/libusb/libusb/libusbi.h

33.193 USBRoster Class Reference

```
#include <haiku_usb.h>
```

Public Member Functions

- USBRoster ()
- virtual ~USBRoster ()
- int Start ()
- void Stop ()

Private Attributes

- void * fLooper

33.193.1 Constructor & Destructor Documentation

USBRoster ( )

~USBRoster ( ) `[virtual]`

33.193.2 Member Function Documentation

int Start ( )

void Stop ( )

33.193.3 Field Documentation

void* fLooper `[private]`

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

- lib/ndkbuild/libusb/libusb/os/haiku_usb.h
- lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp

33.194 USBTransfer Class Reference

```
#include <haiku_usb.h>
```

Public Member Functions

- USBTransfer (struct usbi_transfer *, USBDevice *)
- virtual ~USBTransfer ()
- void Do (int)
- struct usbi_transfer * UsbiTransfer ()
- void SetCancelled ()
- bool IsCancelled ()

Private Attributes

- struct usbi_transfer * fUsbiTransfer
- struct libusb_transfer * fLibusbTransfer
- USBDevice * fUSBDevice
- BLocker fStatusLock
- bool fCancelled

33.194.1 Constructor & Destructor Documentation

USBTransfer (struct **usbi_transfer** * *itransfer,* **USBDevice** * *device*)

~**USBTransfer** ( ) [virtual]

33.194.2 Member Function Documentation

void **Do** (int *fRawFD*)

struct **usbi_transfer** * **UsbiTransfer** ( )

void **SetCancelled** ( )

bool **IsCancelled** ( )

33.194.3 Field Documentation

struct **usbi_transfer*** **fUsbiTransfer** [private]

struct **libusb_transfer*** **fLibusbTransfer** [private]

USBDevice* **fUSBDevice** [private]

BLocker fStatusLock [private]

bool **fCancelled** [private]

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

- lib/ndkbuild/libusb/libusb/os/haiku_usb.h
- lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp

33.195 ushort_array Struct Reference

```
#include <AIOTypes.h>
```

Data Fields

- unsigned size

33.195.1 Field Documentation

unsigned size

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

- lib/AIOTypes.h

33.196 UShortArray Class Reference

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

Public Member Functions

- UShortArray (int size=0)

33.196.1 Constructor & Destructor Documentation

UShortArray (int *size* = 0) [inline]

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

- deprecated/classlib/USBDeviceBase.hpp

33.197 WatchedEntry Class Reference

Public Member Functions

- WatchedEntry (BMessenger ∗, entry_ref ∗)
- ∼WatchedEntry ()
- bool EntryCreated (entry_ref ∗ref)
- bool EntryRemoved (ino_t node)
- bool InitCheck ()

Private Attributes

- BMessenger ∗ fMessenger
- node_ref fNode
- bool fIsDirectory
- USBDevice ∗ fDevice
- WatchedEntry ∗ fEntries
- WatchedEntry ∗ fLink
- bool fInitCheck

33.197.1 Constructor & Destructor Documentation

WatchedEntry (BMessenger ∗ *messenger,* entry_ref ∗ *ref*)

∼WatchedEntry ( )

33.197.2 Member Function Documentation

bool EntryCreated (entry_ref ∗ *ref*)

bool EntryRemoved (ino_t *node*)

bool InitCheck ( )

33.197.3 Field Documentation

BMessenger∗ fMessenger `[private]`

node_ref fNode `[private]`

bool fIsDirectory `[private]`

USBDevice∗ fDevice `[private]`

WatchedEntry∗ fEntries `[private]`

WatchedEntry∗ fLink `[private]`

bool fInitCheck `[private]`

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

- lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp

33.198 wince_device_priv Struct Reference

```
#include <wince_usb.h>
```

Data Fields

- UKW_DEVICE dev
- UKW_DEVICE_DESCRIPTOR desc

33.198.1 Field Documentation

UKW_DEVICE dev

UKW_DEVICE_DESCRIPTOR desc

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

- lib/ndkbuild/libusb/libusb/os/wince_usb.h

33.199 wince_transfer_priv Struct Reference

```
#include <wince_usb.h>
```

Data Fields

- struct winfd pollable_fd
- uint8_t interface_number

33.199.1 Field Documentation

struct winfd pollable_fd

uint8_t interface_number

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

- lib/ndkbuild/libusb/libusb/os/wince_usb.h

33.200 windows_device_handle_priv Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- int active_interface
- struct interface_handle_t interface_handle [USB_MAXINTERFACES]
- int autoclaim_count [USB_MAXINTERFACES]

33.200.1 Field Documentation

int active_interface

struct interface_handle_t interface_handle[USB_MAXINTERFACES]

int autoclaim_count[USB_MAXINTERFACES]

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.201 windows_device_priv Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- uint8_t depth
- uint8_t port
- uint8_t active_config
- struct libusb_device * parent_dev

- struct windows_usb_api_backend
  const * apib
- char * path
- int sub_api
- struct {
  char * path
  struct windows_usb_api_backend
  const * apib
  int sub_api
  int8_t nb_endpoints
  uint8_t * endpoint
  bool restricted_functionality
  } usb_interface [USB_MAXINTERFACES]

- struct hid_device_priv * hid
- USB_DEVICE_DESCRIPTOR dev_descriptor
- unsigned char ** config_descriptor

33.201.1 Field Documentation

uint8_t depth

uint8_t port

uint8_t active_config

struct libusb_device* parent_dev

struct windows_usb_api_backend const* apib

char* path

int sub_api

int8_t nb_endpoints

uint8_t* endpoint

bool restricted_functionality

struct { ... } usb_interface[USB_MAXINTERFACES]

struct hid_device_priv* hid

USB_DEVICE_DESCRIPTOR dev_descriptor

unsigned char config_descriptor**

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.202 windows_transfer_priv Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- struct winfd pollable_fd
- uint8_t interface_number
- uint8_t * hid_buffer
- uint8_t * hid_dest
- size_t hid_expected_size

33.202.1 Field Documentation

struct winfd pollable_fd

uint8_t interface_number

uint8_t* hid_buffer

uint8_t* hid_dest

size_t hid_expected_size

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.203 windows_usb_api_backend Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- const uint8_t id
- const char * designation
- const char ** driver_name_list
- const uint8_t nb_driver_names
- int(* init)(int sub_api, struct libusb_context *ctx)
- int(* exit)(int sub_api)
- int(* open)(int sub_api, struct libusb_device_handle *dev_handle)
- void(* close)(int sub_api, struct libusb_device_handle *dev_handle)
- int(* configure_endpoints)(int sub_api, struct libusb_device_handle *dev_handle, int iface)
- int(* claim_interface)(int sub_api, struct libusb_device_handle *dev_handle, int iface)
- int(* set_interface_altsetting)(int sub_api, struct libusb_device_handle *dev_handle, int iface, int altsetting)
- int(* release_interface)(int sub_api, struct libusb_device_handle *dev_handle, int iface)
- int(* clear_halt)(int sub_api, struct libusb_device_handle *dev_handle, unsigned char endpoint)
- int(* reset_device)(int sub_api, struct libusb_device_handle *dev_handle)
- int(* submit_bulk_transfer)(int sub_api, struct usbi_transfer *itransfer)
- int(* submit_iso_transfer)(int sub_api, struct usbi_transfer *itransfer)
- int(* submit_control_transfer)(int sub_api, struct usbi_transfer *itransfer)
- int(* abort_control)(int sub_api, struct usbi_transfer *itransfer)
- int(* abort_transfers)(int sub_api, struct usbi_transfer *itransfer)
- int(* copy_transfer_data)(int sub_api, struct usbi_transfer *itransfer, uint32_t io_size)

33.203.1 Field Documentation

const uint8_t id

const char* designation

const char driver_name_list**

const uint8_t nb_driver_names

int(* init)(int sub_api, struct libusb_context *ctx)

int(* exit)(int sub_api)

int(* open)(int sub_api, struct libusb_device_handle *dev_handle)

void(* close)(int sub_api, struct libusb_device_handle *dev_handle)

int(* configure_endpoints)(int sub_api, struct libusb_device_handle *dev_handle, int iface)

int(* claim_interface)(int sub_api, struct libusb_device_handle *dev_handle, int iface)

int(* set_interface_altsetting)(int sub_api, struct libusb_device_handle *dev_handle, int iface, int altsetting)

int(∗ release_interface)(int sub_api, struct libusb_device_handle ∗dev_handle, int iface)

int(∗ clear_halt)(int sub_api, struct libusb_device_handle ∗dev_handle, unsigned char endpoint)

int(∗ reset_device)(int sub_api, struct libusb_device_handle ∗dev_handle)

int(∗ submit_bulk_transfer)(int sub_api, struct usbi_transfer ∗itransfer)

int(∗ submit_iso_transfer)(int sub_api, struct usbi_transfer ∗itransfer)

int(∗ submit_control_transfer)(int sub_api, struct usbi_transfer ∗itransfer)

int(∗ abort_control)(int sub_api, struct usbi_transfer ∗itransfer)

int(∗ abort_transfers)(int sub_api, struct usbi_transfer ∗itransfer)

int(∗ copy_transfer_data)(int sub_api, struct usbi_transfer ∗itransfer, uint32_t io_size)

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.204 winfd Struct Reference

```
#include <poll_windows.h>
```

Data Fields

- int fd
- HANDLE handle
- OVERLAPPED ∗ overlapped
- struct usbi_transfer ∗ itransfer
- cancel_transfer ∗ cancel_fn
- enum rw_type rw

33.204.1 Field Documentation

int fd

HANDLE handle

OVERLAPPED∗ overlapped

struct usbi_transfer∗ itransfer

cancel_transfer∗ cancel_fn

enum rw_type rw

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

- lib/ndkbuild/libusb/libusb/os/poll_windows.h

33.205 winusb_interface Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- bool initialized
- WinUsb_AbortPipe_t AbortPipe
- WinUsb_ControlTransfer_t ControlTransfer
- WinUsb_FlushPipe_t FlushPipe
- WinUsb_Free_t Free

- WinUsb_GetAssociatedInterface_t GetAssociatedInterface
- WinUsb_GetCurrentAlternateSetting_t GetCurrentAlternateSetting
- WinUsb_GetDescriptor_t GetDescriptor
- WinUsb_GetOverlappedResult_t GetOverlappedResult
- WinUsb_GetPipePolicy_t GetPipePolicy
- WinUsb_GetPowerPolicy_t GetPowerPolicy
- WinUsb_Initialize_t Initialize
- WinUsb_QueryDeviceInformation_t QueryDeviceInformation
- WinUsb_QueryInterfaceSettings_t QueryInterfaceSettings
- WinUsb_QueryPipe_t QueryPipe
- WinUsb_ReadPipe_t ReadPipe
- WinUsb_ResetPipe_t ResetPipe
- WinUsb_SetCurrentAlternateSetting_t SetCurrentAlternateSetting
- WinUsb_SetPipePolicy_t SetPipePolicy
- WinUsb_SetPowerPolicy_t SetPowerPolicy
- WinUsb_WritePipe_t WritePipe
- WinUsb_ResetDevice_t ResetDevice

33.205.1 Field Documentation

bool initialized

WinUsb_AbortPipe_t AbortPipe

WinUsb_ControlTransfer_t ControlTransfer

WinUsb_FlushPipe_t FlushPipe

WinUsb_Free_t Free

WinUsb_GetAssociatedInterface_t GetAssociatedInterface

WinUsb_GetCurrentAlternateSetting_t GetCurrentAlternateSetting

WinUsb_GetDescriptor_t GetDescriptor

WinUsb_GetOverlappedResult_t GetOverlappedResult

WinUsb_GetPipePolicy_t GetPipePolicy

WinUsb_GetPowerPolicy_t GetPowerPolicy

WinUsb_Initialize_t Initialize

WinUsb_QueryDeviceInformation_t QueryDeviceInformation

WinUsb_QueryInterfaceSettings_t QueryInterfaceSettings

WinUsb_QueryPipe_t QueryPipe

WinUsb_ReadPipe_t ReadPipe

WinUsb_ResetPipe_t ResetPipe

WinUsb_SetCurrentAlternateSetting_t SetCurrentAlternateSetting

WinUsb_SetPipePolicy_t SetPipePolicy

WinUsb_SetPowerPolicy_t SetPowerPolicy

WinUsb_WritePipe_t WritePipe

WinUsb_ResetDevice_t ResetDevice

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.206 WINUSB_PIPE_INFORMATION Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- USBD_PIPE_TYPE PipeType
- UCHAR PipeId
- USHORT MaximumPacketSize
- UCHAR Interval

33.206.1 Field Documentation

USBD_PIPE_TYPE PipeType

UCHAR PipeId

USHORT MaximumPacketSize

UCHAR Interval

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

33.207 WINUSB_SETUP_PACKET Struct Reference

```
#include <windows_winusb.h>
```

Data Fields

- UCHAR request_type
- UCHAR request
- USHORT value
- USHORT index
- USHORT length

33.207.1 Field Documentation

UCHAR request_type

UCHAR request

USHORT value

USHORT index

USHORT length

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

- lib/ndkbuild/libusb/libusb/os/windows_winusb.h

Chapter 34

File Documentation

34.1 deprecated/classlib/AI16_DataPoint.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include "AI16_DataPoint.hpp"
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.2 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

34.3 deprecated/classlib/AI16_DataSet.cpp File Reference

```
#include <iomanip>
#include <assert.h>
#include "AI16_DataSet.hpp"
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.4 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

34.5 deprecated/classlib/AI16_InputRange.cpp File Reference

```
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.6 deprecated/classlib/AI16_InputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class AI16_InputRange

Namespaces

- AIOUSB

34.7 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

34.7.1 Detailed Description

class AnalogInputSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

34.8 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

34.9 deprecated/classlib/AnalogIORange.cpp File Reference

```
#include <assert.h>
#include <math.h>
#include "AnalogIORange.hpp"
```

Namespaces

- AIOUSB

34.10 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

34.11 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

34.12 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

34.13 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

34.14 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

34.15 deprecated/classlib/AO16_OutputRange.cpp File Reference

```
#include "AO16_OutputRange.hpp"
#include "AO16_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.16 deprecated/classlib/AO16_OutputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class AO16_OutputRange

Namespaces

- AIOUSB

34.17 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

34.17.1 Detailed Description

class Counter implementation

Author

Format:

an <ae>

Date

Format:

ad

34.18 deprecated/classlib/Counter.hpp File Reference

```
#include <vector>
```

Data Structures

- class Counter

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

Namespaces

- AIOUSB

34.19 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

34.19.1 Detailed Description

class CounterSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

34.20 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

34.21 deprecated/classlib/CppCommon.h File Reference

34.22 deprecated/classlib/DA12_AnalogOutputSubsystem.cpp File Reference

class DA12_AnalogOutputSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <aiousb.h>
#include "USBDeviceManager.hpp"
#include "DA12_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.22.1 Detailed Description

class DA12_AnalogOutputSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

34.23 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

34.24 deprecated/classlib/DA12_OutputRange.cpp File Reference

```
#include "DA12_OutputRange.hpp"
#include "DA12_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

34.25 deprecated/classlib/DA12_OutputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class DA12_OutputRange

Namespaces

- AIOUSB

34.26 deprecated/classlib/DeviceSubsystem.cpp File Reference

```
#include "DeviceSubsystem.hpp"
#include <assert.h>
```

Namespaces

- AIOUSB

34.27 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

34.28 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

34.28.1 Detailed Description

class DigitalIOSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

34.29 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

34.30 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

34.30.1 Detailed Description

class DIOStreamSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

34.31 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

34.32 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

34.33 deprecated/classlib/README.doc File Reference

34.34 lib/wrappers/README.doc File Reference

34.35 Firmware/README.doc File Reference

34.36 samples/USB-AI16-16/README.doc File Reference

34.37 samples/USB-AO16-16/README.doc File Reference

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

34.39 samples/USB-DIO-16/README.doc File Reference

34.40 samples/USB-DIO-32/README.doc File Reference

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

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

34.43 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

34.44 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

34.45 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

34.46 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

34.47 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

34.48 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

34.49 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

34.50 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

34.51 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

34.52 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

34.53 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

34.54 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

34.55 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

34.56 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

34.57 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

34.58 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

34.59 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

34.60 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

34.61 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)

34.62 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)

34.63 deprecated/classlib/USBDeviceManager.cpp File Reference

class USBDeviceManager implementation

```
#include "CppCommon.h"
#include <iostream>
#include <iterator>
#include <bits/stl_algo.h>
#include <assert.h>
#include <AIOUSB_Core.h>
#include "AIODeviceTable.h"
#include "USBDeviceManager.hpp"
#include "USB_AI16_Family.hpp"
#include "USB_AO16_Family.hpp"
#include "USB_CTR_15_Family.hpp"
#include "USB_DA12_8A_Family.hpp"
#include "USB_DA12_8E_Family.hpp"
#include "USB_DIO_16_Family.hpp"
#include "USB_DIO_32_Family.hpp"
#include "USB_DIO_Family.hpp"
#include "USB_AIO16_Family.hpp"
```

Namespaces

- AIOUSB

34.63.1 Detailed Description

class USBDeviceManager implementation

Author

Format:

an <ae>

Date

Format:

ad

34.64 deprecated/classlib/USBDeviceManager.hpp File Reference

class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations

```
#include "CppCommon.h"
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
#include <aiousb.h>
#include <USBDeviceBase.hpp>
```

Data Structures

- class USBDeviceManager

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

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

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

Namespaces

- AIOUSB

34.64.1 Detailed Description

class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations

Author

Format:

an <ae>

Date

Format:

ad

34.65 doc/aiousb.doc File Reference

34.66 doc/firmware.doc File Reference

34.67 doc/index.doc File Reference

34.68 doc/install.doc File Reference

34.69 doc/java.doc File Reference

34.70 doc/libusb.doc File Reference

34.71 doc/samples.doc File Reference

34.72 doc/wrappers.doc File Reference

34.73 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)

34.73.1 Function Documentation

AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗ *to,* ADCConfigBlock ∗ *from*)

AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗ *config*)

AIOUSBDevice∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗ *obj,* AIORET_TYPE ∗ *result*)

AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗ *obj,* AIOUSBDevice ∗ *dev*)

AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗ *obj,* AIOUSBDevice ∗ *dev*)

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗ *config,* AIOUSBDevice ∗ *dev*)

initializes an ADCConfigBlock using parameters from the AIOUSBDevice

AIORET_TYPE ADCConfigBlockSetSize (**ADCConfigBlock** * *obj,* unsigned *size*)

AIORET_TYPE ADCConfigBlockGetSize (**const ADCConfigBlock** * *obj*)

AIORET_TYPE ADCConfigBlockSetTesting (**ADCConfigBlock** * *obj,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE ADCConfigBlockSetDebug (**ADCConfigBlock** * *obj,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE ADCConfigBlockSetRangeSingle (**ADCConfigBlock** * *config,* unsigned long *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetRegister (**ADCConfigBlock** * *config,* unsigned *reg,* unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetTesting (**const ADCConfigBlock** * *obj*)

AIORET_TYPE ADCConfigBlockGetDebug (**const ADCConfigBlock** * *obj*)

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

Parameters

config	
deviceDesc	
size	

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

Parameters

config	
deviceDesc	

Returns

void ADC_VerifyAndCorrectConfigBlock (**ADCConfigBlock** * *configBlock,* **AIOUSBDevice** * *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (**ADCConfigBlock** * *config,* unsigned *gainCode,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE ADCConfigBlockGetGainCode (**const ADCConfigBlock** * *config,* unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (**ADCConfigBlock** * *config,* unsigned *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetEndChannel (**ADCConfigBlock** * *config,* unsigned char *endChannel*)

AIORET_TYPE ADCConfigBlockSetChannelRange (**ADCConfigBlock** * *config,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

INTERNAL_DOCUMENTATION.

AIORET_TYPE ADCConfigBlockSetStartChannel (**ADCConfigBlock** * *config,* unsigned char *startChannel*)

See Also

USB_SOFTWARE_MANUAL

AIORET_TYPE ADCConfigBlockSetScanRange (**ADCConfigBlock** * *config,* unsigned *startChannel,* unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

AIORET_TYPE ADCConfigBlockSetCalMode (**ADCConfigBlock** * *config,* **ADCalMode** *calMode*)

AIORET_TYPE ADCConfigBlockGetCalMode (**const ADCConfigBlock** * *config*)

AIORET_TYPE ADCConfigBlockGetStartChannel (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetEndChannel (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetOversample (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetOversample (**ADCConfigBlock** ∗ *config,* unsigned *overSample*)

AIORET_TYPE ADCConfigBlockGetTimeout (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTimeout (**ADCConfigBlock** ∗ *config,* unsigned *timeout*)

AIORET_TYPE ADCConfigBlockGetTriggerMode (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTriggerMode (**ADCConfigBlock** ∗ *config,* unsigned *triggerMode*)

AIORET_TYPE ADCConfigBlockSetDifferentialMode (**ADCConfigBlock** ∗ *config,* unsigned *channel,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE ADCConfigBlockSetReference (**ADCConfigBlock** ∗ *config,* int *ref*)

Sets the Timer reference.

AIORET_TYPE ADCConfigBlockSetTriggerEdge (**ADCConfigBlock** ∗ *config,* **AIOUSB_BOOL** *val*)

const char∗ get_gain_code (int *code*)

const char∗ get_cal_mode (int *code*)

char∗ ADCConfigBlockToYAML (**ADCConfigBlock** ∗ *config*)

```
* ---
* adcconfig:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - 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*)

34.74 lib/ndkbuild/jni/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)

34.74.1 Function Documentation

AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock * *to*, ADCConfigBlock * *from*)

AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock * *config*)

AIOUSBDevice* ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock * *obj*, AIORET_TYPE * *result*)

AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock * *obj*, AIOUSBDevice * *dev*)

AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock * *obj*, AIOUSBDevice * *dev*)

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock * *config*)

AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock * *config*, AIOUSBDevice * *dev*)

initializes an ADCConfigBlock using parameters from the AIOUSBDevice

AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock * *obj*, unsigned *size*)

AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock * *obj*)

AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock * *obj*, AIOUSB_BOOL *testing*)

AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock * *obj*, AIOUSB_BOOL *debug*)

AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock * *config*, unsigned long *channel*, unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock * *config*, unsigned *reg*, unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock * *obj*)

AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock * *obj*)

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

Parameters

config	
deviceDesc	
size	

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

Parameters

config	

deviceDesc	

Returns

void ADC_VerifyAndCorrectConfigBlock (**ADCConfigBlock** ∗ *configBlock,* **AIOUSBDevice** ∗ *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (**ADCConfigBlock** ∗ *config,* unsigned *gainCode,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE ADCConfigBlockGetGainCode (**const ADCConfigBlock** ∗ *config,* unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (**ADCConfigBlock** ∗ *config,* unsigned *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetEndChannel (**ADCConfigBlock** ∗ *config,* unsigned char *endChannel*)

AIORET_TYPE ADCConfigBlockSetChannelRange (**ADCConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

INTERNAL_DOCUMENTATION.

AIORET_TYPE ADCConfigBlockSetStartChannel (**ADCConfigBlock** ∗ *config,* unsigned char *startChannel*)

See Also

USB_SOFTWARE_MANUAL

AIORET_TYPE ADCConfigBlockSetScanRange (**ADCConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

AIORET_TYPE ADCConfigBlockSetCalMode (**ADCConfigBlock** ∗ *config,* **ADCalMode** *calMode*)

AIORET_TYPE ADCConfigBlockGetCalMode (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetStartChannel (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetEndChannel (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetOversample (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetOversample (**ADCConfigBlock** ∗ *config,* unsigned *overSample*)

AIORET_TYPE ADCConfigBlockGetTimeout (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTimeout (**ADCConfigBlock** ∗ *config,* unsigned *timeout*)

AIORET_TYPE ADCConfigBlockGetTriggerMode (**const ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTriggerMode (**ADCConfigBlock** ∗ *config,* unsigned *triggerMode*)

AIORET_TYPE ADCConfigBlockSetDifferentialMode (**ADCConfigBlock** ∗ *config,* unsigned *channel,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE ADCConfigBlockSetReference (**ADCConfigBlock** ∗ *config,* int *ref*)

Sets the Timer reference.

AIORET_TYPE ADCConfigBlockSetTriggerEdge (**ADCConfigBlock** ∗ *config,* **AIOUSB_BOOL** *val*)

const char∗ get_gain_code (int *code*)

const char∗ get_cal_mode (int *code*)

char* ADCConfigBlockToYAML (ADCConfigBlock * *config*)

```
* ---
* adcconfig:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - 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*)

34.75 lib/ADCConfigBlock.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct mux_settings
- struct ADCConfigBlock

Typedefs

- typedef struct AIOUSBDevice AIOUSBDevice
- typedef struct mux_settings ADCMuxSettings
- typedef struct ADCConfigBlock ADCConfigBlock
- typedef ADCConfigBlock ADConfigBlock

Functions

- AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock *, AIOUSBDevice *deviceDesc, unsigned int)
- AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock *config, AIOUSBDevice *deviceDesc)
- AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock *config)
- void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock *configBlock, AIOUSBDevice *deviceDesc)
- AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock *config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock *config, unsigned reg, unsigned char value)
- AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock *config, unsigned channel)
- AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock *config, unsigned channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock *config, int clock_rate)
- AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock *config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock *config, unsigned char startChannel)
- AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock *config, unsigned char endChannel)
- AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock *config, unsigned startChannel, unsigned endChannel, unsigned gainCode)

  INTERNAL_DOCUMENTATION.
- AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock *config, ADCalMode calMode)
- AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock *config)
- char * ADCConfigBlockToYAML (ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock *config, unsigned overSample)
- AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock *config, unsigned timeout)
- AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock *config, unsigned triggerMode)
- AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock *config, int ref)

  Sets the Timer reference.
- AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock *config, AIOUSB_BOOL val)
- AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock *config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock *config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock *to, ADCConfigBlock *from)
- AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock *obj, AIOUSBDevice *dev)
- AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock *obj, AIOUSBDevice *dev)
- AIOUSBDevice * ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock *obj, AIORET_TYPE *res)
- AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock *config, AIOUSBDevice *dev)

  initializes an ADCConfigBlock using parameters from the AIOUSBDevice
- AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock *obj, AIOUSB_BOOL testing)
- AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock *obj)
- AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock *obj, unsigned size)
- AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock *obj)
- AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock *obj, AIOUSB_BOOL debug)
- AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock *obj)
- char * ADCConfigBlockToJSON (ADCConfigBlock *config)

  INTERNAL_DOCUMENTATION.
- ADCConfigBlock * NewADCConfigBlockFromJSON (const char *str)
- AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock *config)
- AIOUSB_BOOL is_all_digits (char *str)

34.75.1 Typedef Documentation

typedef struct AIOUSBDevice AIOUSBDevice

typedef struct mux_settings ADCMuxSettings

typedef struct ADCConfigBlock ADCConfigBlock

typedef ADCConfigBlock ADConfigBlock

34.75.2 Function Documentation

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

Parameters

config	
deviceDesc	
size	

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

Parameters

config	
deviceDesc	

Returns

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock * *config*)

void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock * *configBlock*, AIOUSBDevice * *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock * *config*, unsigned *gainCode*, AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock * *config*, unsigned *reg*, unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock * *config*, unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock * *config*, unsigned *channel*, unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock * *config*, int *clock_rate*)

AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock * *config*)

AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock * *config*, unsigned *startChannel*, unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

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*)

34.76 lib/ndkbuild/jni/ADCConfigBlock.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct mux_settings
- struct ADCConfigBlock

Typedefs

- typedef struct AIOUSBDevice AIOUSBDevice
- typedef struct mux_settings ADCMuxSettings
- typedef struct ADCConfigBlock ADCConfigBlock
- typedef ADCConfigBlock ADConfigBlock

Functions

- AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock ∗, AIOUSBDevice ∗deviceDesc, unsigned int)
- AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock ∗config, AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗config)
- void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock ∗configBlock, AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock ∗config, unsigned reg, unsigned char value)
- AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock ∗config, unsigned channel)
- AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock ∗config, unsigned channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock ∗config, int clock_rate)
- AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock ∗config, unsigned char startChannel)
- AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock ∗config, unsigned char endChannel)
- AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel, unsigned gainCode)

  INTERNAL_DOCUMENTATION.
- AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock ∗config, ADCalMode calMode)
- AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock ∗config)
- char ∗ ADCConfigBlockToYAML (ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock ∗config, unsigned overSample)
- AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock ∗config, unsigned timeout)
- AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock ∗config, unsigned triggerMode)
- AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock ∗config, int ref)

  Sets the Timer reference.
- AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock ∗config, AIOUSB_BOOL val)

- AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock ∗config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗to, ADCConfigBlock ∗from)
- AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIOUSBDevice ∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗obj, AIORET_TYPE ∗res)
- AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗config, AIOUSBDevice ∗dev)

  initializes an ADCConfigBlock using parameters from the AIOUSBDevice
- AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock ∗obj, AIOUSB_BOOL testing)
- AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock ∗obj, unsigned size)
- AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock ∗obj, AIOUSB_BOOL debug)
- AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock ∗obj)
- char ∗ ADCConfigBlockToJSON (ADCConfigBlock ∗config)

  INTERNAL_DOCUMENTATION.
- ADCConfigBlock ∗ NewADCConfigBlockFromJSON (const char ∗str)
- AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗config)
- AIOUSB_BOOL is_all_digits (char ∗str)

34.76.1 Typedef Documentation

typedef struct AIOUSBDevice AIOUSBDevice

typedef struct mux_settings ADCMuxSettings

typedef struct ADCConfigBlock ADCConfigBlock

typedef ADCConfigBlock ADConfigBlock

34.76.2 Function Documentation

AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock ∗ *config*, AIOUSBDevice ∗ *deviceDesc*, unsigned *size*)

Parameters

config	
deviceDesc	
size	

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

Parameters

config	
deviceDesc	

Returns

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗ *config*)

void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock ∗ *configBlock*, AIOUSBDevice ∗ *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock ∗ *config*, unsigned *gainCode*, AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock ∗ *config*, unsigned *reg*, unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock ∗ *config*, unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock ∗ *config*, unsigned *channel*, unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock * *config,* int *clock_rate*)

AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock * *config*)

AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock * *config,* unsigned *startChannel,* unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

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
*   - 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*)

34.77 lib/AIOBuf.c File Reference

```
#include "AIOBuf.h"
#include <stdio.h>
#include <stdlib.h>
```

Functions

- AIOBuf ∗ NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetSize (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetTotalSize (AIOBuf ∗buf)
- AIOBufType AIOBufGetType (AIOBuf ∗buf)
- void ∗ AIOBufGetRaw (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetTypeSize (AIOBuf ∗buf)
- AIORET_TYPE AIOBufRead (AIOBuf ∗buf, void ∗tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf ∗buf, void ∗frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf ∗buf)
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗biter)

- void AIOBufIteratorNext (AIOBufIterator *biter)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator *biter)

  Returns a value from the current interator.

34.77.1 Function Documentation

AIOBuf* NewAIOBuf (AIOBufType *type*, size_t *size*)

AIORET_TYPE DeleteAIOBuf (AIOBuf * *buf*)

AIORET_TYPE AIOBufGetSize (AIOBuf * *buf*)

AIORET_TYPE AIOBufGetTotalSize (AIOBuf * *buf*)

AIOBufType AIOBufGetType (AIOBuf * *buf*)

void* AIOBufGetRaw (AIOBuf * *buf*)

AIORET_TYPE AIOBufGetTypeSize (AIOBuf * *buf*)

AIORET_TYPE AIOBufRead (AIOBuf * *buf*, void * *tobuf*, size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (AIOBuf * *buf*, void * *frombuf*, size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (AIOBuf * *buf*)

AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator * *biter*)

void AIOBufIteratorNext (AIOBufIterator * *biter*)

AIOEither AIOBufIteratorGetValue (AIOBufIterator * *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

34.78 lib/ndkbuild/jni/AIOBuf.c File Reference

```
#include "AIOBuf.h"
#include <stdio.h>
#include <stdlib.h>
```

Functions

- AIOBuf * NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf *buf)
- AIORET_TYPE AIOBufGetSize (AIOBuf *buf)
- AIORET_TYPE AIOBufGetTotalSize (AIOBuf *buf)
- AIOBufType AIOBufGetType (AIOBuf *buf)
- void * AIOBufGetRaw (AIOBuf *buf)
- AIORET_TYPE AIOBufGetTypeSize (AIOBuf *buf)
- AIORET_TYPE AIOBufRead (AIOBuf *buf, void *tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf *buf, void *frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf *buf)
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator *biter)
- void AIOBufIteratorNext (AIOBufIterator *biter)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator *biter)

  Returns a value from the current interator.

34.78.1 Function Documentation

AIOBuf* NewAIOBuf (**AIOBufType** *type,* size_t *size*)

AIORET_TYPE DeleteAIOBuf (**AIOBuf** * *buf*)

AIORET_TYPE AIOBufGetSize (**AIOBuf** * *buf*)

AIORET_TYPE AIOBufGetTotalSize (**AIOBuf** * *buf*)

AIOBufType AIOBufGetType (**AIOBuf** * *buf*)

void* AIOBufGetRaw (**AIOBuf** * *buf*)

AIORET_TYPE AIOBufGetTypeSize (**AIOBuf** * *buf*)

AIORET_TYPE AIOBufRead (**AIOBuf** * *buf,* **void** * *tobuf,* size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (**AIOBuf** * *buf,* **void** * *frombuf,* size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (**AIOBuf** * *buf*)

AIOUSB_BOOL AIOBufIteratorIsValid (**AIOBufIterator** * *biter*)

void AIOBufIteratorNext (**AIOBufIterator** * *biter*)

AIOEither AIOBufIteratorGetValue (**AIOBufIterator** * *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

34.79 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,
  AIO_ERROR_BUF = -1, AIO_DEFAULT_BUF = 1, AIO_COUNTS_BUF = 2, AIO_VOLTS_BUF = 8 }

Functions

- AIOBuf * NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf *type)
- AIORET_TYPE AIOBufGetSize (AIOBuf *buf)
- AIOBufType AIOBufGetType (AIOBuf *buf)
- void * AIOBufGetRaw (AIOBuf *buf)
- AIORET_TYPE AIOBufRead (AIOBuf *buf, void *tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf *buf, void *frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf *buf)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator *biter)

  Returns a value from the current interator.
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator *biter)
- void AIOBufIteratorNext (AIOBufIterator *biter)

34.79.1 Typedef Documentation

typedef struct AIOBuf AIOBuf

typedef struct aiobuf_iterator AIOBufIterator

34.79.2 Enumeration Type Documentation

enum AIOBufType

Enumerator

AIO_ERROR_BUF

AIO_DEFAULT_BUF

AIO_COUNTS_BUF

AIO_VOLTS_BUF

AIO_ERROR_BUF

AIO_DEFAULT_BUF

AIO_COUNTS_BUF

AIO_VOLTS_BUF

34.79.3 Function Documentation

AIOBuf* NewAIOBuf (AIOBufType *type*, size_t *size*)

AIORET_TYPE DeleteAIOBuf (AIOBuf * *type*)

AIORET_TYPE AIOBufGetSize (AIOBuf * *buf*)

AIOBufType AIOBufGetType (AIOBuf * *buf*)

void* AIOBufGetRaw (AIOBuf * *buf*)

AIORET_TYPE AIOBufRead (AIOBuf * *buf*, void * *tobuf*, size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (AIOBuf * *buf*, void * *frombuf*, size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (AIOBuf * *buf*)

AIOEither AIOBufIteratorGetValue (AIOBufIterator * *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗ *biter*)

void AIOBufIteratorNext (AIOBufIterator ∗ *biter*)

34.80 lib/ndkbuild/jni/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,
  AIO_ERROR_BUF = -1, AIO_DEFAULT_BUF = 1, AIO_COUNTS_BUF = 2, AIO_VOLTS_BUF = 8 }

Functions

- AIOBuf ∗ NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf ∗type)
- AIORET_TYPE AIOBufGetSize (AIOBuf ∗buf)
- AIOBufType AIOBufGetType (AIOBuf ∗buf)
- void ∗ AIOBufGetRaw (AIOBuf ∗buf)
- AIORET_TYPE AIOBufRead (AIOBuf ∗buf, void ∗tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf ∗buf, void ∗frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf ∗buf)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator ∗biter)

  Returns a value from the current interator.
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗biter)
- void AIOBufIteratorNext (AIOBufIterator ∗biter)

34.80.1 Typedef Documentation

typedef struct **AIOBuf AIOBuf**

typedef struct **aiobuf_iterator AIOBufIterator**

34.80.2 Enumeration Type Documentation

enum **AIOBufType**

Enumerator

AIO_ERROR_BUF

AIO_DEFAULT_BUF

AIO_COUNTS_BUF

AIO_VOLTS_BUF

AIO_ERROR_BUF

AIO_DEFAULT_BUF

AIO_COUNTS_BUF

AIO_VOLTS_BUF

34.80.3 Function Documentation

AIOBuf∗ NewAIOBuf (**AIOBufType** *type,* size_t *size*)

AIORET_TYPE DeleteAIOBuf (**AIOBuf** ∗ *type*)

AIORET_TYPE AIOBufGetSize (**AIOBuf** ∗ *buf*)

AIOBufType AIOBufGetType (**AIOBuf** ∗ *buf*)

void∗ AIOBufGetRaw (**AIOBuf** ∗ *buf*)

AIORET_TYPE AIOBufRead (**AIOBuf** ∗ *buf,* **void** ∗ *tobuf,* size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (**AIOBuf** ∗ *buf,* **void** ∗ *frombuf,* size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (**AIOBuf** ∗ *buf*)

AIOEither AIOBufIteratorGetValue (**AIOBufIterator** ∗ *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator * *biter*)

void AIOBufIteratorNext (AIOBufIterator * *biter*)

34.81 lib/AIOChannelMask.c File Reference

```
#include "AIOChannelMask.h"
#include "AIOTypes.h"
#include <stdio.h>
#include <string.h>
```

Functions

- AIOChannelMask * NewAIOChannelMask (unsigned number_channels)

  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask *mask)

  Destructor for the AIOChannelMask object.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask *mask, int *pos)

  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask *mask, int *pos)

  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask *obj, unsigned field)

  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask *obj, char field, unsigned index)

  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask *obj, char *tmp, unsigned index)

  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask *obj)

  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask *obj)

  *Returns channels that are set to High (not low) *.*
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask *obj)
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask *obj, const char *bitfields)

  Rely on the base type to determine the sizes.
- AIOChannelMask * NewAIOChannelMaskFromStr (const char *bitfields)

  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask * NewAIOChannelMaskFromChr (const char bits)
- char * AIOChannelMaskToString (AIOChannelMask *obj)

  Returns a string representation for the AIOChannel Bit mask in question.
- char * AIOChannelMaskToStringAtIndex (AIOChannelMask *obj, unsigned index)

  Returns a mask for the index in question.
- char * AIOChannelMaskGetMask (AIOChannelMask *obj)

34.81.1 Function Documentation

AIOChannelMask* NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask * *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

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

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask * *mask,* int * *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask * *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask * *obj,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask * *obj,* char * *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
**tmp*	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask * *obj*)

Gives the size of the given BitMask.

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask * *obj*)

Returns channels that are set to High (not low) *.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask * *obj*)

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

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

AIOChannelMask* NewAIOChannelMaskFromStr (const char * *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ NewAIOChannelMaskFromChr (const char *bits*)

char∗ AIOChannelMaskToString (AIOChannelMask ∗ *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

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

char∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗ *obj,* unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

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

char∗ AIOChannelMaskGetMask (AIOChannelMask ∗ *obj*)

34.82 lib/ndkbuild/jni/AIOChannelMask.c File Reference

```
#include "AIOChannelMask.h"
#include "AIOTypes.h"
#include <stdio.h>
#include <string.h>
```

Functions

- AIOChannelMask ∗ NewAIOChannelMask (unsigned number_channels)

  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)

  Destructor for the AIOChannelMask object.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)

  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)

  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗obj, unsigned field)

  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗obj, char field, unsigned index)

  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗obj, char ∗tmp, unsigned index)

  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗obj)

  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗obj)

  Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗obj)

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

  Rely on the base type to determine the sizes.
- AIOChannelMask * NewAIOChannelMaskFromStr (const char *bitfields)

  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask * NewAIOChannelMaskFromChr (const char bits)
- char * AIOChannelMaskToString (AIOChannelMask *obj)

  Returns a string representation for the AIOChannel Bit mask in question.
- char * AIOChannelMaskToStringAtIndex (AIOChannelMask *obj, unsigned index)

  Returns a mask for the index in question.
- char * AIOChannelMaskGetMask (AIOChannelMask *obj)

34.82.1 Function Documentation

AIOChannelMask* NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask * *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

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

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask * *mask,* int * *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask * *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask * *obj,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask * *obj,* char * *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
**tmp*	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask * *obj*)

Gives the size of the given BitMask.

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask * *obj*)

Returns channels that are set to High (not low) *.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask * *obj*)

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

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

AIOChannelMask* NewAIOChannelMaskFromStr (const char * *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask* NewAIOChannelMaskFromChr (const char *bits*)

char* AIOChannelMaskToString (AIOChannelMask * *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

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

char* AIOChannelMaskToStringAtIndex (AIOChannelMask * *obj*, unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

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

char* AIOChannelMaskGetMask (AIOChannelMask * *obj*)

34.83 lib/AIOChannelMask.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
```

Data Structures

- struct AIOChannelMask

Macros

- #define BIT_LENGTH(x) (sizeof(x) ∗ 8)

Typedefs

- typedef char aio_channel_obj

Functions

- AIOChannelMask ∗ NewAIOChannelMask (unsigned size)
  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)
  Destructor for the AIOChannelMask object.
- AIOChannelMask ∗ NewAIOChannelMaskFromStr (const char ∗bitfields)
  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask ∗ NewAIOChannelMaskFromChr (const char bits)
- char ∗ AIOChannelMaskToString (AIOChannelMask ∗mask)
  Returns a string representation for the AIOChannel Bit mask in question.
- char ∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗obj, unsigned index)
  Returns a mask for the index in question.
- char ∗ AIOChannelMaskGetMask (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗mask, char ∗val, unsigned index)
  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗mask)
  Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗mask)
  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)
  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)
  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗mask, unsigned field)
  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗mask, char field, unsigned index)
  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗mask, const char ∗bitfields)
  Rely on the base type to determine the sizes.

34.83.1 Macro Definition Documentation

#define BIT_LENGTH(*x*) (sizeof(x) ∗ 8)

34.83.2 Typedef Documentation

typedef char aio_channel_obj

34.83.3 Function Documentation

AIOChannelMask∗ NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask ∗ *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

AIOChannelMask∗ NewAIOChannelMaskFromStr (const char ∗ *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ NewAIOChannelMaskFromChr (const char *bits*)

char∗ AIOChannelMaskToString (AIOChannelMask ∗ *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

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

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

char∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗ *obj,* unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

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

char∗ AIOChannelMaskGetMask (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗ *obj,* char ∗ *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
∗tmp	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗ *obj*)

Returns channels that are set to High (not low) ∗.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗ *mask*)

Gives the size of the given BitMask.

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

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗ *mask,* int ∗ *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗ *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗ *mask,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

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

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

34.84 lib/ndkbuild/jni/AIOChannelMask.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
```

Data Structures

- struct AIOChannelMask

Macros

- #define BIT_LENGTH(x) (sizeof(x) ∗ 8)

Typedefs

- typedef char aio_channel_obj

Functions

- AIOChannelMask ∗ NewAIOChannelMask (unsigned size)

  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)

  Destructor for the AIOChannelMask object.
- AIOChannelMask ∗ NewAIOChannelMaskFromStr (const char ∗bitfields)

  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask ∗ NewAIOChannelMaskFromChr (const char bits)
- char ∗ AIOChannelMaskToString (AIOChannelMask ∗mask)

  Returns a string representation for the AIOChannel Bit mask in question.
- char ∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗obj, unsigned index)

  Returns a mask for the index in question.
- char ∗ AIOChannelMaskGetMask (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗mask, char ∗val, unsigned index)

  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗mask)

  Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗mask)

  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)

  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)

  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗mask, unsigned field)

  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗mask, char field, unsigned index)

  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗mask, const char ∗bitfields)

  Rely on the base type to determine the sizes.

34.84.1 Macro Definition Documentation

#define BIT_LENGTH(*x*) (sizeof(x) ∗ 8)

34.84.2 Typedef Documentation

typedef char aio_channel_obj

34.84.3 Function Documentation

AIOChannelMask∗ NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask ∗ *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

AIOChannelMask∗ NewAIOChannelMaskFromStr (const char ∗ *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ NewAIOChannelMaskFromChr (const char *bits*)

char∗ AIOChannelMaskToString (AIOChannelMask ∗ *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

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

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

char∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗ *obj,* unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

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

char∗ AIOChannelMaskGetMask (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗ *obj,* char ∗ *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
∗*tmp*	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗ *obj*)

Returns channels that are set to High (not low) ∗.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask * *mask*)

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask * *mask*)

Gives the size of the given BitMask.

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

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask * *mask,* int * *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask * *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask * *mask,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

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

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

34.85 lib/AIOChannelRange.c File Reference

```
#include "AIOChannelRange.h"
```

Enumerations

- enum STATE {
  BEGIN, START_CHANNEL, END_CHANNEL, GAIN,
  END, 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

34.85.1 Enumeration Type Documentation

enum STATE

Enumerator

BEGIN

START_CHANNEL

END_CHANNEL

GAIN

END

BEGIN

START_CHANNEL

END_CHANNEL

GAIN

END

34.85.2 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*)

34.85.3 Variable Documentation

int aio_channel_range_error = 0

34.86 lib/ndkbuild/jni/AIOChannelRange.c File Reference

```
#include "AIOChannelRange.h"
```

Enumerations

- enum STATE {
  BEGIN, START_CHANNEL, END_CHANNEL, GAIN,
  END, 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

34.86.1 Enumeration Type Documentation

enum STATE

Enumerator

BEGIN

START_CHANNEL

END_CHANNEL

GAIN

END

BEGIN

START_CHANNEL

END_CHANNEL

GAIN

END

34.86.2 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*)

34.86.3 Variable Documentation

int aio_channel_range_error = 0

34.87 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

34.87.1 Macro Definition Documentation

#define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain_pairs) - 1))

34.87.2 Typedef Documentation

typedef struct aio_channel_range AIOChannelRange

34.87.3 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*)

34.87.4 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

34.88 lib/ndkbuild/jni/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

34.88.1 Macro Definition Documentation

#define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain_pairs) - 1))

34.88.2 Typedef Documentation

typedef struct aio_channel_range AIOChannelRange

34.88.3 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*)

34.88.4 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

34.89 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)

34.89.1 Function Documentation

AIOCmd* NewAIOCmdFromJSON (const char * *str*)

AIOCmd* NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (AIOCmd * *cmd*)

34.90 lib/ndkbuild/jni/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)

34.90.1 Function Documentation

AIOCmd* NewAIOCmdFromJSON (const char * *str*)

AIOCmd* NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (AIOCmd * *cmd*)

34.91 lib/AIOCmd.h File Reference

```
#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)

34.91.1 Typedef Documentation

typedef struct AIOCmd AIOCmd

34.91.2 Function Documentation

AIOCmd∗ NewAIOCmdFromJSON (const char ∗ *str*)

AIOCmd∗ NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (AIOCmd ∗ *cmd*)

34.92 lib/ndkbuild/jni/AIOCmd.h File Reference

```
#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)

34.92.1 Typedef Documentation

typedef struct AIOCmd AIOCmd

34.92.2 Function Documentation

AIOCmd∗ NewAIOCmdFromJSON (const char ∗ *str*)

AIOCmd∗ NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (AIOCmd ∗ *cmd*)

34.93 lib/AIOCommandLine.c File Reference

```
#include "AIOCommandLine.h"
#include "AIOList.h"
```

Functions

- AIOCommandLineOptions ∗ AIO_CMDLINE_DEFAULT_OPTIONS ()
- AIOCommandLineOptions ∗ AIO_CMDLINE_SCRIPTING_OPTIONS ()
- AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗opts)
- AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions ∗options, int ∗argc, char ∗∗argv)
- AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions ∗options, int argc, char ∗∗argv)

  A simplified command line parsing library for.
- void AIOPrintUsage (int argc, char ∗∗argv, struct option ∗options)

  Shows the user the various options that this library is capable of parsing on the command line.
- AIOCommandLineOptions ∗ NewDefaultAIOCommandLineOptions ()

  Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

- AIOCommandLineOptions * NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions *orig)
- AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions *options)

  A Descructor for the allocated AIOCommandLineOptions.
- AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions *options, int *indices, int num_devices)
- AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions *options, intlist *indices)

  Lists any devices that were matched and then lists which index was specified.
- AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock *config, AIOCommandLineOptions *options)
- AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock *config, AIOCommandLineOptions *options)

  Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.
- AIOChannelRangeTmp * AIOGetChannelRange (char *optarg)
- const AIOCommandLineOptions * AIO_SCRIPTING_OPTIONS (void)
- const AIOCommandLineOptions * AIO_CMDLINE_OPTIONS (void)

Variables

- int opterr
- int optind
- AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

  The default settings for running various samples.
- AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

34.93.1 Function Documentation

AIOCommandLineOptions* AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions* AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions * *opts*)

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

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

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

A simplified command line parsing library for.

Parameters

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

Returns

void AIOPrintUsage (int *argc*, char ** *argv*, struct option * *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIOCommandLineOptions* NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions * a new object containing the default settings

AIOCommandLineOptions* NewAIOCommandLineOptionsFromDefaultOptions (**const AIOCommandLineOptions** * *orig*)

AIORET_TYPE DeleteAIOCommandLineOptions (**AIOCommandLineOptions** * *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

AIORET_TYPE AIOCommandLineListDevices (**AIOCommandLineOptions** * *options,* int * *indices,* int *num_devices*)

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

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

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

>= AIOUSB_SUCCESS if devices have been found, < 0 if no devices found

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (**ADCConfigBlock** * *config,* **AIOCommandLineOptions** * *options*)

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (**ADCConfigBlock** * *config,* **AIOCommandLineOptions** * *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

>= AIOUSB_SUCCESS is successful, < 0 if there was an error

AIOChannelRangeTmp* AIOGetChannelRange (char * *optarg*)

const AIOCommandLineOptions* AIO_SCRIPTING_OPTIONS (**void**)

const AIOCommandLineOptions* AIO_CMDLINE_OPTIONS (void)

34.93.2 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

34.94 lib/ndkbuild/jni/AIOCommandLine.c File Reference

```
#include "AIOCommandLine.h"
#include "AIOList.h"
```

Functions

- AIOCommandLineOptions * AIO_CMDLINE_DEFAULT_OPTIONS ()
- AIOCommandLineOptions * AIO_CMDLINE_SCRIPTING_OPTIONS ()
- AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions *opts)
- AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions *options, int *argc, char **argv)
- AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions *options, int argc, char **argv)

  A simplified command line parsing library for.
- void AIOPrintUsage (int argc, char **argv, struct option *options)

  Shows the user the various options that this library is capable of parsing on the command line.
- AIOCommandLineOptions * NewDefaultAIOCommandLineOptions ()

  Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.
- AIOCommandLineOptions * NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions *orig)
- AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions *options)

  A Descructor for the allocated AIOCommandLineOptions.
- AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions *options, int *indices, int num_devices)
- AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions *options, intlist *indices)

  Lists any devices that were matched and then lists which index was specified.
- AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock *config, AIOCommandLineOptions *options)
- AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock *config, AIOCommandLineOptions *options)

  Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.
- AIOChannelRangeTmp * AIOGetChannelRange (char *optarg)
- const AIOCommandLineOptions * AIO_SCRIPTING_OPTIONS (void)
- const AIOCommandLineOptions * AIO_CMDLINE_OPTIONS (void)

Variables

- int opterr
- int optind
- AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

  The default settings for running various samples.
- AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

34.94.1 Function Documentation

AIOCommandLineOptions* AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions* AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (**AIOCommandLineOptions** * *opts*)

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

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

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

A simplified command line parsing library for.

Parameters

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

Returns

void AIOPrintUsage (int *argc,* char ** *argv,* struct **option** * *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIOCommandLineOptions* NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions * a new object containing the default settings

AIOCommandLineOptions* NewAIOCommandLineOptionsFromDefaultOptions (**const AIOCommandLineOptions** * *orig*)

AIORET_TYPE DeleteAIOCommandLineOptions (**AIOCommandLineOptions** * *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

AIORET_TYPE AIOCommandLineListDevices (**AIOCommandLineOptions** * *options,* int * *indices,* int *num_devices*)

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

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

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

>= AIOUSB_SUCCESS if devices have been found, < 0 if no devices found

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock * *config,* AIOCommandLineOptions * *options*)

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock * *config,* AIOCommandLineOptions * *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

>= AIOUSB_SUCCESS is successful, < 0 if there was an error

AIOChannelRangeTmp* AIOGetChannelRange (char * *optarg*)

const AIOCommandLineOptions* AIO_SCRIPTING_OPTIONS (void)

const AIOCommandLineOptions* AIO_CMDLINE_OPTIONS (void)

34.94.2 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

34.95 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, 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

34.95.1 Macro Definition Documentation

#define DUMP 0x1000

#define CNTS 0x1001

#define JCONF 0x1002

#define REPEAT 0x1003

34.95.2 Typedef Documentation

typedef struct **AIOChannelRangeTmp AIOChannelRangeTmp**

typedef struct **AIOCommandLineOptions AIOCommandLineOptions**

34.95.3 Enumeration Type Documentation

enum **DeviceEnum**

Enumerator

INDEX_NUM
ADCCONFIG_OPT
TIMEOUT_OPT
DEBUG_OPT
SETCAL_OPT
COUNT_OPT
SAMPLE_OPT
FILE_OPT
CHANNEL_OPT
INDEX_NUM
ADCCONFIG_OPT
TIMEOUT_OPT
DEBUG_OPT
SETCAL_OPT
COUNT_OPT
SAMPLE_OPT
FILE_OPT
CHANNEL_OPT

34.95.4 Function Documentation

AIOCommandLineOptions∗ NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions ∗ a new object containing the default settings

AIOCommandLineOptions∗ NewAIOCommandLineOptionsFromDefaultOptions (**const AIOCommandLineOptions** ∗ *orig*)

AIOCommandLineOptions∗ AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions∗ AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (**AIOCommandLineOptions** ∗ *opts*)

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

A simplified command line parsing library for.

Parameters

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

Returns

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

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

AIOChannelRangeTmp* AIOGetChannelRange (char * *optarg*)

void AIOPrintUsage (int *argc*, char ** *argv*, struct option * *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions * *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

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

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

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

>= AIOUSB_SUCCESS if devices have been found, < 0 if no devices found

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock * *config*, AIOCommandLineOptions * *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

>= AIOUSB_SUCCESS is successful, < 0 if there was an error

const AIOCommandLineOptions* AIO_SCRIPTING_OPTIONS (void)

const AIOCommandLineOptions* AIO_CMDLINE_OPTIONS (void)

AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions * *options,* int * *indices,* int *num_devices*)

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock * *config,* AIOCommandLineOptions * *options*)

34.95.5 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

34.96 lib/ndkbuild/jni/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, 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

34.96.1 Macro Definition Documentation

#define DUMP 0x1000

#define CNTS 0x1001

#define JCONF 0x1002

#define REPEAT 0x1003

34.96.2 Typedef Documentation

typedef struct AIOChannelRangeTmp AIOChannelRangeTmp

typedef struct AIOCommandLineOptions AIOCommandLineOptions

34.96.3 Enumeration Type Documentation

enum DeviceEnum

Enumerator

INDEX_NUM
ADCCONFIG_OPT
TIMEOUT_OPT
DEBUG_OPT
SETCAL_OPT
COUNT_OPT
SAMPLE_OPT
FILE_OPT
CHANNEL_OPT
INDEX_NUM
ADCCONFIG_OPT
TIMEOUT_OPT
DEBUG_OPT
SETCAL_OPT
COUNT_OPT
SAMPLE_OPT
FILE_OPT
CHANNEL_OPT

34.96.4 Function Documentation

AIOCommandLineOptions∗ NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions ∗ a new object containing the default settings

AIOCommandLineOptions∗ NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions ∗ *orig*)

AIOCommandLineOptions∗ AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions∗ AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗ *opts*)

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

A simplified command line parsing library for.

Parameters

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

Returns

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

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

AIOChannelRangeTmp* **AIOGetChannelRange** (**char** * *optarg*)

void AIOPrintUsage (**int** *argc,* **char** ** *argv,* **struct option** * *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIORET_TYPE DeleteAIOCommandLineOptions (**AIOCommandLineOptions** * *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

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

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

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

>= AIOUSB_SUCCESS if devices have been found, < 0 if no devices found

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (**ADCConfigBlock** * *config,* **AIOCommandLineOptions** * *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

>= AIOUSB_SUCCESS is successful, < 0 if there was an error

const AIOCommandLineOptions* **AIO_SCRIPTING_OPTIONS** (**void**)

const AIOCommandLineOptions* **AIO_CMDLINE_OPTIONS** (**void**)

AIORET_TYPE AIOCommandLineListDevices (**AIOCommandLineOptions** * *options,* **int** * *indices,* **int** *num_devices*)

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock * *config,* AIOCommandLineOptions * *options*)

34.96.5 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

34.97 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)

34.97.1 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*)

34.98 lib/ndkbuild/jni/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)

34.98.1 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*)

34.99 lib/AIOConfiguration.h File Reference

```
#include "AIOTypes.h"
#include "AIOContinuousBuffer.h"
#include "ADCConfigBlock.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
```

Data Structures

- struct configuration
- struct AIOArgument
- struct AIOArguments

Typedefs

- typedef struct configuration AIOConfiguration
- typedef AIOConfiguration ADCConfiguration
- typedef AIOConfiguration AIOContinousBufConfiguration

Enumerations

- enum ConfigurationType {
  NO_CONFIG = -1, ADCCONIGBLOCK_CONFIG = 0, AIOCONTINUOUSBUF_CONFIG = 1, 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, 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, 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)

34.99.1 Typedef Documentation

typedef struct configuration AIOConfiguration

typedef AIOConfiguration ADCConfiguration

typedef AIOConfiguration AIOContinousBufConfiguration

34.99.2 Enumeration Type Documentation

enum ConfigurationType

Enumerator

NO_CONFIG
ADCCONIGBLOCK_CONFIG
AIOCONTINUOUSBUF_CONFIG
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
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
AD_NO_SET_CAL
AD_SET_CAL_AUTO
AD_SET_CAL_NORMAL
AD_SET_CAL_MANUAL

34.99.3 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*)

34.100 lib/ndkbuild/jni/AIOConfiguration.h File Reference

```
#include "AIOTypes.h"
#include "AIOContinuousBuffer.h"
#include "ADCConfigBlock.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
```

Data Structures

- struct configuration
- struct AIOArgument
- struct AIOArguments

Typedefs

- typedef struct configuration AIOConfiguration
- typedef AIOConfiguration ADCConfiguration
- typedef AIOConfiguration AIOContinousBufConfiguration

Enumerations

- enum ConfigurationType {
  NO_CONFIG = -1, ADCCONIGBLOCK_CONFIG = 0, AIOCONTINUOUSBUF_CONFIG = 1, 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, 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, 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)

34.100.1 Typedef Documentation

typedef struct configuration AIOConfiguration

typedef AIOConfiguration ADCConfiguration

typedef AIOConfiguration AIOContinousBufConfiguration

34.100.2 Enumeration Type Documentation

enum ConfigurationType

Enumerator

NO_CONFIG

ADCCONIGBLOCK_CONFIG

AIOCONTINUOUSBUF_CONFIG

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

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

AD_NO_SET_CAL

AD_SET_CAL_AUTO

AD_SET_CAL_NORMAL

AD_SET_CAL_MANUAL

34.100.3 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*)

34.101 lib/AIOContinuousBuffer.c File Reference

```
#include "AIOUSB_Log.h"
#include "AIOContinuousBuffer.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "AIOChannelMask.h"
#include "AIOUSB_CTR.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include "AIOFifo.h"
#include "AIOCountsConverter.h"
#include "AIOCmd.h"
#include "cJSON.h"
#include <ctype.h>
```

Data Structures

- struct rangelookup

Typedefs

- typedef struct rangelookup RangeValueLookup

Functions

- void * ConvertCountsToVoltsFunction (void *object)

  Main work function for collecting data.
- void * RawCountsWorkFunction (void *object)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusitionOverrun (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusition (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf * NewAIOContinuousBuf (unsigned long deviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf *buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf *buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBuf_InitConfiguration (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf *buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf *buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBuf_SendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf *buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf *buf, unsigned blksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf *buf)
- ADCConfigBlock * AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_SetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf *buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetSizeNumElements (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf *buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf *buf)

  returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufCountsAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetDataAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, unsigned size)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, int64_t num_scans)

will read in an integer number of scan counts if there is room.

- AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf *buf, AIOBuf *tobuf, size_t size_to_read)
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned read_buf_size)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf *buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf *buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf *buf)

  Starts the thread that acquires data from USB bus.
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf *buf, AIOChannelMask *mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBuf_NumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_NumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf *buf, AIOBufferType *writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

  Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf *buf, unsigned short *data, unsigned 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(*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 readbufsize, 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 []

34.101.1 Typedef Documentation

typedef struct rangelookup RangeValueLookup

34.101.2 Function Documentation

void * ConvertCountsToVoltsFunction (void * *object*)

Main work function for collecting data.

Also performs copies from the raw acquiring buffer into the AIOContinuousBuf

Parameters

object	

Returns

Todo Ensure that copying matches the actual size of the data

Also performs copies from the raw acquiring buffer into the AIOContinuousBuf

Parameters

object	

Returns

Todo Ensure that copying matches the actual size of the data

create temporary buffer and then Load the fifo with values

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

void ∗ **RawCountsWorkFunction** (**void** ∗ *object*)

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

AIORET_TYPE **AIOContinuousBufForceTerminateAcqusitionOverrun** (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE **AIOContinuousBufForceTerminateAcqusition** (**AIOContinuousBuf** ∗ *buf*)

AIOContinuousBuf∗ **NewAIOContinuousBufForCounts** (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels*)

AIOContinuousBuf∗ **NewAIOContinuousBuf** (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_-oversamples	
base_size	

Returns

AIORET_TYPE **AIOContinuousBufGetNumberChannels** (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE **AIOContinuousBufSetNumberChannels** (**AIOContinuousBuf** ∗ *buf,* unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE **AIOContinuousBufSetBaseSize** (**AIOContinuousBuf** ∗ *buf,* size_t *newbase*)

AIORET_TYPE **AIOContinuousBufGetBaseSize** (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE **AIOContinuousBufGetBufferSize** (**AIOContinuousBuf** ∗ *buf*)

AIOContinuousBuf∗ **NewAIOContinuousBufForVolts** (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels,* unsigned *num_oversamples*)

AIORET_TYPE **AIOContinuousBuf_InitConfiguration** (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE **AIOContinuousBufPushN** (**AIOContinuousBuf** ∗ *buf,* **void** ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE **AIOContinuousBufPopN** (**AIOContinuousBuf** ∗ *buf,* **void** ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE **AIOContinuousBufInitADCConfigBlock** (**AIOContinuousBuf** ∗ *buf,* unsigned *size,* **ADGainCode** *gainCode,* **AIOUSB_BOOL** *diffMode,* unsigned char *os,* **AIOUSB_BOOL** *dfs*)

AIORET_TYPE **AIOContinuousBufInitConfiguration** (**AIOContinuousBuf** ∗ *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value < 0 if not.

AIORET_TYPE AIOContinuousBuf_SendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE DeleteAIOContinuousBuf (**AIOContinuousBuf** ∗ *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufSetCountsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (**AIOContinuousBuf** ∗ *buf,* unsigned *blksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (**AIOContinuousBuf** ∗ *buf*)

ADCConfigBlock∗ AIOContinuousBufGetADCConfigBlock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetCallback (**AIOContinuousBuf** ∗ *buf,* **void** ∗(∗)(**void** ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetCallback (**AIOContinuousBuf** ∗ *buf,* **void** ∗(∗)(**void** ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetNumberScans (**AIOContinuousBuf** ∗ *buf,* **int64_t** *num_scans*)

AIORET_TYPE AIOContinuousBufGetNumberScans (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetUnitSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (**AIOContinuousBuf** ∗ *buf,* **uint16_t** *new_unit_size*)

AIORET_TYPE AIOContinuousBufReset (**AIOContinuousBuf** ∗ *buf*)

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufGetReadPosition (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetSizeNumElements (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufPending (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (**AIOContinuousBuf** ∗ *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (**AIOContinuousBuf** ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufCountsAvailable (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetDataAvailable (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

•

Parameters

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf * *buf*, int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf * *buf*, unsigned short * *read_buf*, unsigned *tmpbuffer_size*, int64_t *num_scans*)

will read in an integer number of scan counts if there is room.

Parameters

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf * *buf*, AIOBuf * *tobuf*, size_t *size_to_read*)

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf * *buf*, unsigned short * *read_buf*, unsigned *read_buf_size*)

AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf * *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf * *buf*, unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf * *buf*)

Starts the thread that acquires data from USB bus.

Parameters

buf	

Returns

status code of start.

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf * *buf*)

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

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBuf_NumberSignals (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSignals (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_NumberChannels (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberChannels (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufWrite (**AIOContinuousBuf** ∗ *buf,* **AIOBufferType** ∗ *writebuf,* unsigned *wrbufsize,* unsigned *size,* **AIOContinuousBufMode** *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *data,* unsigned *datasize,* unsigned *size,* **AIOContinuousBufMode** *flag*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE StartStreaming (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE SetConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE ResetCounters (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufLoadCounters (**AIOContinuousBuf** ∗ *buf,* unsigned *countera,* unsigned *counterb*)

AIORET_TYPE AIOContinuousBufCleanup (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufPreSetup (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadNSamples (**AIOContinuousBuf** ∗ *buf,* **void** ∗ *tobuf,* size_t *n_to_read*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (**AIOContinuousBuf** ∗ *buf*)

unsigned long number_to_read (**AIOContinuousBuf** ∗ *buf,* **AIOCmd** ∗ *cmd*)

AIOUSB_BOOL continue_running (**AIOContinuousBuf** ∗ *buf,* **AIOCmd** ∗ *cmd*)

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf * *buf,* AIOCmd * *cmd,* AIORET_TYPE(*)(AIOContinuousBuf *buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf * *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf * *buf,* AIOBufferType * *readbuf,* unsigned *readbufsize,* unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure *.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

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

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf * *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gainCode*)

AIORET_TYPE AIOContinuousBufEnd (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufSetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDefaultModeForCounterScan (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBuf_SaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBuf_SetChannelRangeGain (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBuf_SetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetTimeout (**AIOContinuousBuf** ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOverSample (**AIOContinuousBuf** ∗ *buf,* size_t *os*)

AIORET_TYPE AIOContinuousBuf_GetOverSample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBuf_SetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBuf_GetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

cJSON* GetJSONValueOrDefault (**cJSON** * *config,* char **const** * *key,* **EnumStringLookup** * *lookup,* size_t *size*)

AIOContinuousBuf* NewAIOContinuousBufFromJSON (**const** char * *str*)

char* AIOContinuousBufToJSON (**AIOContinuousBuf** * *buf*)

34.101.3 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 }
}
```

34.102 lib/ndkbuild/jni/AIOContinuousBuffer.c File Reference

```
#include "AIOUSB_Log.h"
#include "AIOContinuousBuffer.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "AIOChannelMask.h"
#include "AIOUSB_CTR.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include "AIOFifo.h"
#include "AIOCountsConverter.h"
#include "AIOCmd.h"
#include "cJSON.h"
#include <ctype.h>
```

Data Structures

- struct rangelookup

Typedefs

- typedef struct rangelookup RangeValueLookup

Functions

- void * ConvertCountsToVoltsFunction (void *object)

  Main work function for collecting data.
- void * RawCountsWorkFunction (void *object)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusitionOverrun (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusition (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf * NewAIOContinuousBuf (unsigned long deviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf *buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf *buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBuf_InitConfiguration (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf *buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf *buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBuf_SendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf *buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf *buf, unsigned blksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf *buf)
- ADCConfigBlock * AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_SetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf *buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetSizeNumElements (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf *buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf *buf)

  returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufCountsAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetDataAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, unsigned size)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, int64_t num_scans)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf *buf, AIOBuf *tobuf, size_t size_to_read)
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned read_buf_size)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf *buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf *buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf *buf)

  Starts the thread that acquires data from USB bus.
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf *buf, AIOChannelMask *mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBuf_NumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBuf_NumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf *buf)

- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf *buf, AIOBufferType *writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

  Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf *buf, unsigned short *data, unsigned 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(*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 readbufsize, 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 []

34.102.1 Typedef Documentation

typedef struct rangelookup RangeValueLookup

34.102.2 Function Documentation

void* ConvertCountsToVoltsFunction (void * *object*)

Main work function for collecting data.

Also performs copies from the raw acquiring buffer into the AIOContinuousBuf

Parameters

object	

Returns

Todo Ensure that copying matches the actual size of the data

create temporary buffer and then Load the fifo with values

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

void* RawCountsWorkFunction (void * *object*)

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

AIORET_TYPE AIOContinuousBufForceTerminateAcqusitionOverrun (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufForceTerminateAcqusition (AIOContinuousBuf * *buf*)

AIOContinuousBuf* NewAIOContinuousBufForCounts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels*)

AIOContinuousBuf* NewAIOContinuousBuf (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	

num_-oversamples	
base_size	

Returns

AIORET_TYPE AIOContinuousBufGetNumberChannels (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberChannels (**AIOContinuousBuf** ∗ *buf,* unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE AIOContinuousBufSetBaseSize (**AIOContinuousBuf** ∗ *buf,* size_t *newbase*)

AIORET_TYPE AIOContinuousBufGetBaseSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (**AIOContinuousBuf** ∗ *buf*)

AIOContinuousBuf∗ NewAIOContinuousBufForVolts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels,* unsigned *num_oversamples*)

AIORET_TYPE AIOContinuousBuf_InitConfiguration (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufPushN (**AIOContinuousBuf** ∗ *buf,* **void** ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (**AIOContinuousBuf** ∗ *buf,* **void** ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufInitADCConfigBlock (**AIOContinuousBuf** ∗ *buf,* unsigned *size,* **ADGainCode** *gainCode,* **AIOUSB_BOOL** *diffMode,* unsigned char *os,* **AIOUSB_BOOL** *dfs*)

AIORET_TYPE AIOContinuousBufInitConfiguration (**AIOContinuousBuf** ∗ *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value $<$ 0 if not.

AIORET_TYPE AIOContinuousBuf_SendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE DeleteAIOContinuousBuf (**AIOContinuousBuf** ∗ *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufSetCountsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (**AIOContinuousBuf** ∗ *buf,* unsigned *blksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (**AIOContinuousBuf** ∗ *buf*)

ADCConfigBlock∗ AIOContinuousBufGetADCConfigBlock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetCallback (**AIOContinuousBuf** ∗ *buf,* **void** ∗(∗)(**void** ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetCallback (**AIOContinuousBuf** ∗ *buf,* **void** ∗(∗)(**void** ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf * *buf*, int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf * *buf*, uint16_t *new_unit_size*)

AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf * *buf*)

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetSizeNumElements (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf * *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf * *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufCountsAvailable (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetDataAvailable (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf * *buf*, unsigned short * *read_buf*, unsigned *tmpbuffer_size*, unsigned *size*)

will read in an integer number of scan counts if there is room.

- **Parameters**

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf * *buf*, int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf * *buf*, unsigned short * *read_buf*, unsigned *tmpbuffer_size*, int64_t *num_scans*)

will read in an integer number of scan counts if there is room.

-

Parameters

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf * *buf*, AIOBuf * *tobuf*, size_t *size_to_read*)

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf * *buf*, unsigned short * *read_buf*, unsigned *read_buf_size*)

AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf * *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf * *buf*, unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf * *buf*)

Starts the thread that acquires data from USB bus.

- **Parameters**

buf	

Returns

status code of start.

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf * *buf*)

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

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBuf_NumberSignals (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBuf_NumberChannels (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf * *buf*, AIOBufferType * *writebuf*, unsigned *wrbufsize*, unsigned *size*, AIOContinuousBufMode *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf * *buf,* unsigned short * *data,* unsigned *datasize,* unsigned *size,* AIOContinuousBufMode *flag*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf * *buf*)

AIORET_TYPE StartStreaming (AIOContinuousBuf * *buf*)

AIORET_TYPE SetConfig (AIOContinuousBuf * *buf*)

AIORET_TYPE ResetCounters (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufLoadCounters (AIOContinuousBuf * *buf,* unsigned *countera,* unsigned *counterb*)

AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufPreSetup (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufReadNSamples (AIOContinuousBuf * *buf,* void * *tobuf,* size_t *n_to_read*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf * *buf*)

unsigned long number_to_read (AIOContinuousBuf * *buf,* AIOCmd * *cmd*)

AIOUSB_BOOL continue_running (AIOContinuousBuf * *buf,* AIOCmd * *cmd*)

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf * *buf,* AIOCmd * *cmd,* AIORET_TYPE(*)(AIOContinuousBuf *buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf * *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗ *buf,* AIOBufferType ∗ *readbuf,* unsigned *readbufsize,* unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure ∗.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

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

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗ *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗ *buf,* ADGainCode *gainCode*)

AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetTesting (AIOContinuousBuf ∗ *buf,* AIOUSB_BOOL *testing*)

AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf ∗ *buf,* AIOUSB_BOOL *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDefaultModeForCounterScan (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf ∗ *buf,* AIOUSB_BOOL *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetDeviceIndex (AIOContinuousBuf ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBuf_SaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBuf_SetChannelRangeGain (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBuf_SetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetTimeout (**AIOContinuousBuf** ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOverSample (**AIOContinuousBuf** ∗ *buf,* size_t *os*)

AIORET_TYPE AIOContinuousBuf_GetOverSample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBuf_SetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBuf_GetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

cJSON∗ GetJSONValueOrDefault (**cJSON** ∗ *config,* char **const** ∗ *key,* **EnumStringLookup** ∗ *lookup,* size_t *size*)

AIOContinuousBuf∗ NewAIOContinuousBufFromJSON (**const** char ∗ *str*)

char∗ AIOContinuousBufToJSON (**AIOContinuousBuf** ∗ *buf*)

34.102.3 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 }
}
```

34.103 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, AIO_CONT_BUF_TYPE_COUNTS = 2, AIO_CONT_BUF_TYPE_VOLTS = 8 }

Functions

- AIOContinuousBuf ∗ NewAIOContinuousBuf (unsigned long DeviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIOContinuousBuf ∗ NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf ∗ NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf ∗buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf ∗buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf ∗buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf ∗buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf ∗buf, void ∗(∗work)(void ∗object))
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf ∗buf, unsigned sblksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf ∗buf)
- ADCConfigBlock ∗ AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf ∗buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf ∗buf)

- AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf *buf, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf *buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf *buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf *buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf *buf, ADGainCode gain, AIOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf *buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf *buf, AIOChannelMask *mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf *buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf *buf, unsigned timeout)
- AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf *buf, AIOUSB_BOOL debug)
- AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf *buf, void *tobuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf *buf)

  Setups the Automated runs for continuous mode runs.
- AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf *buf, AIOCmd *cmd, AIORET_TYPE(*callback)(AIOContinuousBuf *buf))

  *Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.*
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf *buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf *buf, unsigned short *tmp, unsigned tmpsize, unsigned size)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned read_buf_size)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, int64_t num_scans)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf *buf)

  returns the number of Scans accross all channels that still remain in the buffer

- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗buf, ADGainCode gainCode)
- AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗buf, AIOBufferType ∗readbuf, unsigned readbufsize, unsigned size)

  Reads the current available amount of data from buf, into the readbuf datastructure ∗.
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗buf, AIOBufferType ∗writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

  Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf ∗buf, unsigned short ∗data, unsigned datasize, unsigned size, AIOContinuousBufMode flag)
- AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf ∗buf)
- char ∗ AIOContinuousBufToJSON (AIOContinuousBuf ∗buf)
- AIOContinuousBuf ∗ NewAIOContinuousBufFromJSON (const char ∗json_string)

34.103.1 Macro Definition Documentation

#define ROOTCLOCK 10000000

34.103.2 Typedef Documentation

typedef void∗(∗ AIOUSB_WorkFn)(void ∗obj)

typedef struct AIOContinuousBuf AIOContinuousBuf

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

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

- Create a new AIOContinuousBuf of a certain size that is large enough to handle the running clock rate ∗ number_of_oversamples ∗
- Assign a device index to the AIOContinuousBuf
- Start am acquisition by calling AIOContinuousBufInitiateCallbackAcquisition;
- Process the input data using either a simple while loop burst_test.c

  or using the callback function as in

34.103.3 Enumeration Type Documentation

enum AIO_CONT_BUF_TYPE

Enumerator

AIO_CONT_BUF_TYPE_COUNTS

AIO_CONT_BUF_TYPE_VOLTS

AIO_CONT_BUF_TYPE_COUNTS

AIO_CONT_BUF_TYPE_VOLTS

34.103.4 Function Documentation

AIOContinuousBuf∗ NewAIOContinuousBuf (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_oversamples	

base_size	

Returns

AIOContinuousBuf* **NewAIOContinuousBufForCounts** (**unsigned long** *DeviceIndex,* **unsigned** *scancounts,* **unsigned** *num_channels*)

AIOContinuousBuf* **NewAIOContinuousBufForVolts** (**unsigned long** *DeviceIndex,* **unsigned** *scancounts,* **unsigned** *num_channels,* **unsigned** *num_oversamples*)

AIORET_TYPE **DeleteAIOContinuousBuf** (**AIOContinuousBuf** * *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE **AIOContinuousBufInitConfiguration** (**AIOContinuousBuf** * *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value < 0 if not.

AIORET_TYPE **AIOContinuousBufInitADCConfigBlock** (**AIOContinuousBuf** * *buf,* **unsigned** *size,* **ADGainCode** *gainCode,* **AIOUSB_BOOL** *diffMode,* **unsigned char** *os,* **AIOUSB_BOOL** *dfs*)

AIOUSB_WorkFn **AIOContinuousBufGetCallback** (**AIOContinuousBuf** * *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE **AIOContinuousBufSetCallback** (**AIOContinuousBuf** * *buf,* **void** *(*)(**void** *object) *work*)

AIORET_TYPE **AIOContinuousBufSetStreamingBlockSize** (**AIOContinuousBuf** * *buf,* **unsigned** *sblksize*)

AIORET_TYPE **AIOContinuousBufGetStreamingBlockSize** (**AIOContinuousBuf** * *buf*)

ADCConfigBlock* **AIOContinuousBufGetADCConfigBlock** (**AIOContinuousBuf** * *buf*)

AIORET_TYPE **AIOContinuousBufSetNumberChannels** (**AIOContinuousBuf** * *buf,* **unsigned** *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE **AIOContinuousBufGetNumberChannels** (**AIOContinuousBuf** * *buf*)

AIORET_TYPE **AIOContinuousBufGetOversample** (**AIOContinuousBuf** * *buf*)

AIORET_TYPE **AIOContinuousBufSetOversample** (**AIOContinuousBuf** * *buf,* **unsigned** *num_oversamples*)

AIORET_TYPE **AIOContinuousBufNumberChannels** (**AIOContinuousBuf** * *buf*)

AIORET_TYPE **AIOContinuousBufSetBaseSize** (**AIOContinuousBuf** * *buf,* **size_t** *newbase*)

AIORET_TYPE **AIOContinuousBufGetBaseSize** (**AIOContinuousBuf** * *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (**AIOContinuousBuf** ∗ *buf,* **uint16_t** *new_unit_size*)

AIORET_TYPE AIOContinuousBufGetUnitSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

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

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBufNumberSignals (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufResetDevice (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTimeout (**AIOContinuousBuf** ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetNumberScans (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberScans (**AIOContinuousBuf** ∗ *buf,* **int64_t** *num_scans*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufReset (**AIOContinuousBuf** ∗ *buf*)

Todo Fix this to use condition variable

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf ∗ *buf,* void ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf ∗ *buf,* void ∗ *tobuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗ *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf ∗ *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗ *buf,* AIOCmd ∗ *cmd,* AIORET_TYPE(∗)(AIOContinuousBuf ∗buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

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

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetReadPosition (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (**AIOContinuousBuf** ∗ *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufPending (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

Parameters

-

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *read_buf_size*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* **int64_t** *num_scans*)

will read in an integer number of scan counts if there is room.

Parameters

-

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufSetCountsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (**AIOContinuousBuf** ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufSetClock (**AIOContinuousBuf** ∗ *buf,* unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufEnd (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf * *buf*, ADGainCode *gainCode*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf * *buf*, AIOBufferType * *readbuf*, unsigned *readbufsize*, unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure *.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

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

AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf * *buf*, AIOBufferType * *writebuf*, unsigned *wrbufsize*, unsigned *size*, AIOContinuousBufMode *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf * *buf*, unsigned short * *data*, unsigned *datasize*, unsigned *size*, AIOContinuousBufMode *flag*)

AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf * *buf*)

char* AIOContinuousBufToJSON (AIOContinuousBuf * *buf*)

AIOContinuousBuf* NewAIOContinuousBufFromJSON (const char * *json_string*)

34.104 lib/ndkbuild/jni/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, AIO_CONT_BUF_TYPE_COUNTS = 2, AIO_CONT_BUF_TYPE_VOLTS = 8 }

Functions

- AIOContinuousBuf * NewAIOContinuousBuf (unsigned long DeviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIOContinuousBuf * NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf * NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf *buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf *buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf *buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf *buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf *buf, unsigned sblksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf *buf)
- ADCConfigBlock * AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf *buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf *buf, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf *buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf *buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf *buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf *buf, ADGainCode gain, AIOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf *buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf *buf, AIOChannelMask *mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf *buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf *buf)

- AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf *buf, unsigned timeout)
- AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf *buf, AIOUSB_BOOL debug)
- AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf *buf, void *tobuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf *buf)

  Setups the Automated runs for continuous mode runs.
- AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf *buf, AIOCmd *cmd, AIORET_TYPE(*callback)(AIOContinuousBuf *buf))

  *Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.*
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf *buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf *buf, unsigned short *tmp, unsigned tmpsize, unsigned size)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned read_buf_size)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, int64_t num_scans)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf *buf)

  returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf *buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf *buf, ADGainCode gainCode)
- AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf *buf, AIOBufferType *readbuf, unsigned readbufsize, unsigned size)

  *Reads the current available amount of data from buf, into the readbuf datastructure *.*
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf *buf, AIOBufferType *writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

  Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf *buf, unsigned short *data, unsigned datasize, unsigned size, AIOContinuousBufMode flag)
- AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf *buf)
- char * AIOContinuousBufToJSON (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufFromJSON (const char *json_string)

34.104.1 Macro Definition Documentation

#define ROOTCLOCK 10000000

34.104.2 Typedef Documentation

typedef void*(* AIOUSB_WorkFn)(void *obj)

typedef struct AIOContinuousBuf AIOContinuousBuf

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

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

- Create a new AIOContinuousBuf of a certain size that is large enough to handle the running clock rate * number-_of_oversamples *
- Assign a device index to the AIOContinuousBuf
- Start am acquisition by calling AIOContinuousBufInitiateCallbackAcquisition;
- Process the input data using either a simple while loop burst_test.c

  or using the callback function as in

34.104.3 Enumeration Type Documentation

enum AIO_CONT_BUF_TYPE

Enumerator

AIO_CONT_BUF_TYPE_COUNTS

AIO_CONT_BUF_TYPE_VOLTS

AIO_CONT_BUF_TYPE_COUNTS

AIO_CONT_BUF_TYPE_VOLTS

34.104.4 Function Documentation

AIOContinuousBuf* NewAIOContinuousBuf (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_-oversamples	
base_size	

Returns

AIOContinuousBuf* NewAIOContinuousBufForCounts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels*)

AIOContinuousBuf* NewAIOContinuousBufForVolts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels,* unsigned *num_oversamples*)

AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf * *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf * *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value < 0 if not.

AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf * *buf*, unsigned *size*, ADGainCode *gainCode*, AIOUSB_BOOL *diffMode*, unsigned char *os*, AIOUSB_BOOL *dfs*)

AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf * *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf * *buf*, void *(*)(void *object) *work*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf * *buf*, unsigned *sblksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf * *buf*)

ADCConfigBlock* AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf * *buf*, unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf * *buf*, unsigned *num_oversamples*)

AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf * *buf*, size_t *newbase*)

AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf * *buf*, uint16_t *new_unit_size*)

AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf * *buf*, AIOUSB_BOOL *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf * *buf*, unsigned *startChannel*, unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf * *buf*, ADGainCode *gain*, AIOUSB_BOOL *diff*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

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

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf * *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf * *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf * *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf * *buf,* AIOUSB_BOOL *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf * *buf,* int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf * *buf*)

Todo Fix this to use condition variable

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf * *buf,* void * *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf * *buf,* void * *tobuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf * *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf * *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗ *buf,* AIOCmd ∗ *cmd,* AIORET_TYPE(∗)(AIOContinuousBuf ∗buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf ∗ *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

-

Parameters

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf ∗ *buf*, unsigned short ∗ *read_buf*, unsigned *read_buf_size*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf ∗ *buf*, unsigned short ∗ *read_buf*, unsigned *tmpbuffer_size*, int64_t *num_scans*)

will read in an integer number of scan counts if there is room.

Parameters

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗ *buf*, unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗ *buf*, ADGainCode *gainCode*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗ *buf*, AIOBufferType ∗ *readbuf*, unsigned *readbufsize*, unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure ∗.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

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

AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗ *buf*, AIOBufferType ∗ *writebuf*, unsigned *wrbufsize*, unsigned *size*, AIOContinuousBufMode *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf * *buf,* unsigned short * *data,* unsigned *datasize,* unsigned *size,* AIOContinuousBufMode *flag*)

AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf * *buf*)

char* AIOContinuousBufToJSON (AIOContinuousBuf * *buf*)

AIOContinuousBuf* NewAIOContinuousBufFromJSON (const char * *json_string*)

34.105 lib/AIOCountsConverter.c File Reference

```
#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)

34.105.1 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*)

34.106 lib/ndkbuild/jni/AIOCountsConverter.c File Reference

```
#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)

34.106.1 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*)

34.107 lib/AIOCountsConverter.h File Reference

```
#include "AIOTypes.h"
#include "ADCConfigBlock.h"
#include "AIOContinuousBuffer.h"
#include "AIOFifo.h"
```

Data Structures

- struct AIOGainRange
- struct aio_counts_converter

Typedefs

- typedef struct aio_counts_converter AIOCountsConverter

Functions

- AIOCountsConverter ∗ NewAIOCountsConverterWithBuffer (void ∗buf, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)

- AIOCountsConverter * NewAIOCountsConverter (unsigned num_channels, AIOGainRange *ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter * NewAIOCountsConverterFromAIOContinuousBuf (void *buf)
- AIOCountsConverter * NewAIOCountsConverterWithScanLimiter (void *buf, unsigned num_scans, unsigned num_channels, AIOGainRange *ranges, unsigned num_oversamples, unsigned unit_size)
- void AIOCountsConverterReset (AIOCountsConverter *cc)
- void DeleteAIOCountsConverter (AIOCountsConverter *ccv)
- AIORET_TYPE AIOCountsConverterConvertNScans (AIOCountsConverter *cc, int num_scans)
- AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (AIOCountsConverter *cc)
- AIORET_TYPE AIOCountsConverterConvert (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIORET_TYPE AIOCountsConverterConvertFifo (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIOGainRange * NewAIOGainRangeFromADCConfigBlock (ADCConfigBlock *adc)
- void DeleteAIOGainRange (AIOGainRange *)

34.107.1 Typedef Documentation

typedef struct **aio_counts_converter AIOCountsConverter**

34.107.2 Function Documentation

AIOCountsConverter* NewAIOCountsConverterWithBuffer (**void** * *buf,* unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter* NewAIOCountsConverter (unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter* NewAIOCountsConverterFromAIOContinuousBuf (**void** * *buf*)

AIOCountsConverter* NewAIOCountsConverterWithScanLimiter (**void** * *buf,* unsigned *num_scans,* unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

void AIOCountsConverterReset (**AIOCountsConverter** * *cc*)

void DeleteAIOCountsConverter (**AIOCountsConverter** * *ccv*)

AIORET_TYPE AIOCountsConverterConvertNScans (**AIOCountsConverter** * *cc,* int *num_scans*)

AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (**AIOCountsConverter** * *cc*)

AIORET_TYPE AIOCountsConverterConvert (**AIOCountsConverter** * *cc,* **void** * *tobuf,* **void** * *frombuf,* unsigned *num_bytes*)

AIORET_TYPE AIOCountsConverterConvertFifo (**AIOCountsConverter** * *cc,* **void** * *tobufptr,* **void** * *frombufptr,* unsigned *num_counts*)

Parameters

cc	Counts converter object
tobufptr	ToFifo (double)
frombufptr	From Fifo (unsigned short)
num_counts	number of counts to convert

Returns

Number of tobufptr objects that have been created

AIOGainRange* NewAIOGainRangeFromADCConfigBlock (**ADCConfigBlock** * *adc*)

void DeleteAIOGainRange (**AIOGainRange** *)

34.108 lib/ndkbuild/jni/AIOCountsConverter.h File Reference

```
#include "AIOTypes.h"
#include "ADCConfigBlock.h"
#include "AIOContinuousBuffer.h"
#include "AIOFifo.h"
```

Data Structures

- struct AIOGainRange
- struct aio_counts_converter

Typedefs

- typedef struct aio_counts_converter AIOCountsConverter

Functions

- AIOCountsConverter * NewAIOCountsConverterWithBuffer (void *buf, unsigned num_channels, AIOGainRange *ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter * NewAIOCountsConverter (unsigned num_channels, AIOGainRange *ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter * NewAIOCountsConverterFromAIOContinuousBuf (void *buf)
- AIOCountsConverter * NewAIOCountsConverterWithScanLimiter (void *buf, unsigned num_scans, unsigned num_channels, AIOGainRange *ranges, unsigned num_oversamples, unsigned unit_size)
- void AIOCountsConverterReset (AIOCountsConverter *cc)
- void DeleteAIOCountsConverter (AIOCountsConverter *ccv)
- AIORET_TYPE AIOCountsConverterConvertNScans (AIOCountsConverter *cc, int num_scans)
- AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (AIOCountsConverter *cc)
- AIORET_TYPE AIOCountsConverterConvert (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIORET_TYPE AIOCountsConverterConvertFifo (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIOGainRange * NewAIOGainRangeFromADCConfigBlock (ADCConfigBlock *adc)
- void DeleteAIOGainRange (AIOGainRange *)

34.108.1 Typedef Documentation

typedef struct **aio_counts_converter AIOCountsConverter**

34.108.2 Function Documentation

AIOCountsConverter* NewAIOCountsConverterWithBuffer (**void** * *buf,* unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter* NewAIOCountsConverter (unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter* NewAIOCountsConverterFromAIOContinuousBuf (**void** * *buf*)

AIOCountsConverter* NewAIOCountsConverterWithScanLimiter (**void** * *buf,* unsigned *num_scans,* unsigned *num_channels,* **AIOGainRange** * *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

void AIOCountsConverterReset (**AIOCountsConverter** * *cc*)

void DeleteAIOCountsConverter (**AIOCountsConverter** * *ccv*)

AIORET_TYPE AIOCountsConverterConvertNScans (**AIOCountsConverter** * *cc,* int *num_scans*)

AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (**AIOCountsConverter** * *cc*)

AIORET_TYPE AIOCountsConverterConvert (**AIOCountsConverter** * *cc,* **void** * *tobuf,* **void** * *frombuf,* unsigned *num_bytes*)

AIORET_TYPE AIOCountsConverterConvertFifo (**AIOCountsConverter** * *cc,* **void** * *tobufptr,* **void** * *frombufptr,* unsigned *num_counts*)

Parameters

cc	Counts converter object
tobufptr	ToFifo (double)

frombufptr	From Fifo (unsigned short)
num_counts	number of counts to convert

Returns

Number of tobufptr objects that have been created

AIOGainRange* **NewAIOGainRangeFromADCConfigBlock** (**ADCConfigBlock** * *adc*)

void DeleteAIOGainRange (**AIOGainRange** *)

34.109 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)

34.109.1 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*)

34.110 lib/ndkbuild/jni/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)

34.110.1 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*)

34.111 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)

34.111.1 Typedef Documentation

typedef struct **AIODeviceInfo** **AIODeviceInfo**

34.111.2 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*)

34.112 lib/ndkbuild/jni/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)

34.112.1 Typedef Documentation

typedef struct AIODeviceInfo AIODeviceInfo

34.112.2 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*)

34.113 lib/AIODeviceQuery.c File Reference

```
#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.

34.113.1 Function Documentation

AIODeviceQuery* **NewAIODeviceQuery (unsigned long *DeviceIndex*)**

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery ∗ object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗ *devq*)

Destructor for AIODeviceQuery ∗.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, success, otherwise error

char∗ AIODeviceQueryToStr (AIODeviceQuery ∗ *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

char∗ AIODeviceQueryToRepr (AIODeviceQuery ∗ *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗ *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

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

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery ∗ *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0 index , otherwise error

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗ *devq*)

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

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the name length in question, otherwise error

char* AIODeviceQueryGetName (AIODeviceQuery * *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery * *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery * *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

34.114 lib/ndkbuild/jni/AIODeviceQuery.c File Reference

```
#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.

34.114.1 Function Documentation

AIODeviceQuery* NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery * object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery * *devq*)

Destructor for AIODeviceQuery *.

Parameters

devq	AIODeviceQuery *

Returns

>= 0, success, otherwise error

char* AIODeviceQueryToStr (AIODeviceQuery * *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

char* AIODeviceQueryToRepr (AIODeviceQuery * *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery * *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery * *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 index , otherwise error

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery * *devq*)

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

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the name length in question, otherwise error

char* AIODeviceQueryGetName (AIODeviceQuery * *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery * *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery * *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

34.115 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.

34.115.1 Typedef Documentation

typedef struct AIODeviceQuery AIODeviceQuery

34.115.2 Function Documentation

AIODeviceQuery∗ NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery ∗ object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗ *devq*)

Destructor for AIODeviceQuery ∗.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, success, otherwise error

char∗ AIODeviceQueryToStr (AIODeviceQuery ∗ *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

char* AIODeviceQueryToRepr (AIODeviceQuery * *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery * *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery * *devq*)

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

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the name length in question, otherwise error

char* AIODeviceQueryGetName (AIODeviceQuery * *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery * *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery * *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery * *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 index , otherwise error

34.116 lib/ndkbuild/jni/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.

34.116.1 Typedef Documentation

typedef struct AIODeviceQuery AIODeviceQuery

34.116.2 Function Documentation

AIODeviceQuery∗ NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery ∗ object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗ *devq*)

Destructor for AIODeviceQuery ∗.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, success, otherwise error

char∗ AIODeviceQueryToStr (AIODeviceQuery ∗ *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

char∗ AIODeviceQueryToRepr (AIODeviceQuery ∗ *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗ *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

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

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗ *devq*)

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

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the name length in question, otherwise error

char* AIODeviceQueryGetName (AIODeviceQuery * *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery * *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery * *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

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

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery * *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 index , otherwise error

34.117 lib/AIODeviceTable.c File Reference

```
#include "AIODeviceTable.h"
#include <string.h>
#include <errno.h>
```

Macros

- #define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

Functions

- AIOUSB_BOOL AIOUSB_SetInit ()
- void AIODeviceTableInit (void)
- AIOUSB_BOOL AIOUSB_IsInit ()
- unsigned long AIOUSB_InitTest (void)
- AIOUSB_BOOL AIOUSB_Cleanup ()
- unsigned long QueryDeviceInfo (unsigned long DeviceIndex, unsigned long *pPID, unsigned long *pNameSize, char *pName, unsigned long *pDIOBytes, unsigned long *pCounters)

  Identifying devices on the USB bus.
- PRIVATE char * ProductIDToName (unsigned int productID)

  this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()
- PRIVATE AIORET_TYPE ProductNameToID (const char *name)
- AIORET_TYPE GetDevices (void)
- USBDevice * AIODeviceTableGetUSBDeviceAtIndex (unsigned long DeviceIndex, AIORESULT *res)
- char * GetSafeDeviceName (unsigned long DeviceIndex)

  GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.
- AIORET_TYPE AIOUSB_EnsureOpen (unsigned long DeviceIndex)
- AIOUSBDevice * AIODeviceTableGetDeviceAtIndex (unsigned long DeviceIndex, AIORESULT *res)
- AIOUSBDevice * AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSBGetError ()
- AIORESULT AIODeviceTableAddDeviceToDeviceTable (int *numAccesDevices, unsigned long productID)

  A mock function that can set up the DeviceTable with any type of devices.
- AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int *numAccesDevices, unsigned long productID, USBDevice *usb_dev)
- AIORET_TYPE ClearAIODeviceTable (int numDevices)

  cleans up the AIODeviceTable and frees any memory associated with it.
- AIORESULT AIODeviceTableSetDeviceID (int index, AIOUSBDevice *dev)
- AIORESULT AIOUSB_GetAllDevices ()
- AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long *products, int length)
- void CloseAllDevices (void)
- unsigned long AIODeviceTableClearDevices (void)
- unsigned long ClearDevices (void)
- AIORET_TYPE AIODeviceTablePopulateTable (void)

  populate device table with ACCES devices found on USB bus
- AIORET_TYPE AIOUSB_Init (void)

  AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.
- AIORET_TYPE AIOUSB_Exit ()
- AIORET_TYPE AIOUSB_Reset (unsigned long DeviceIndex)

Variables

- AIOUSBDevice deviceTable [MAX_USB_DEVICES]
- unsigned long AIOUSB_INIT_PATTERN = 0x9b6773adul
- unsigned long aiousbInit = 0

34.117.1 Macro Definition Documentation

#define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

34.117.2 Function Documentation

AIOUSB_BOOL AIOUSB_SetInit ( )

void AIODeviceTableInit (void)

AIOUSB_BOOL AIOUSB_IsInit ( )

unsigned long AIOUSB_InitTest (void)

AIOUSB_BOOL AIOUSB_Cleanup ( )

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex*, unsigned long * *pPID*, unsigned long * *pNameSize*, char * *pName*, unsigned long * *pDIOBytes*, unsigned long * *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

PRIVATE char* **ProductIDToName** (unsigned int *productID*)

this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()

productIDIndex[] represents an index into productIDNameTable[], sorted by product ID; specifically, it contains pointers into productIDNameTable[]; to get the actual product ID, the pointer in productIDIndex[] must be dereferenced; using a separate index table instead of sorting productIDNameTable[] directly permits us to create multiple indexes, in particular, a second index sorted by product name

PRIVATE AIORET_TYPE **ProductNameToID** (const char * *name*)

This function is the complement of ProductIDToName() and returns the product ID for a given name; this function should be used with care; it will work reliably if passed a name obtained from ProductIDToName(); however, if passed a name obtained from the device itself it may not work; the reason is that devices contain their own name strings, which are most likely identical to the names defined in this module, but not guaranteed to be so; that's not as big a problem as it sounds, however, because if one has the means to obtain the name from the device, then they also have access to the device's product ID, so calling this function is unnecessary; this function is mainly for performing simple conversions between product names and IDs, primarily to support user interfaces

Parameters

name	

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

AIORET_TYPE **GetDevices** (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if < 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid

USBDevice* **AIODeviceTableGetUSBDeviceAtIndex** (unsigned long *DeviceIndex,* **AIORESULT** * *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice * A Usb handle that can be used for USB transactions

char* GetSafeDeviceName (unsigned long *DeviceIndex*)

GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.

AIORET_TYPE AIOUSB_EnsureOpen (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSBDevice* AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

AIOUSBDevice* AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

AIORET_TYPE AIOUSBGetError ( )

AIORESULT AIODeviceTableAddDeviceToDeviceTable (int * *numAccesDevices,* unsigned long *productID*)

A mock function that can set up the DeviceTable with any type of devices.

AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int * *numAccesDevices,* unsigned long *productID,* USBDevice * *usb_dev*)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

AIORESULT AIODeviceTableSetDeviceID (int *index,* AIOUSBDevice * *dev*)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long * *products,* int *length*)

void CloseAllDevices (void)

unsigned long AIODeviceTableClearDevices (void)

unsigned long ClearDevices (void)

AIORET_TYPE AIODeviceTablePopulateTable (void)

populate device table with ACCES devices found on USB bus

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

Note

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

AIORET_TYPE AIOUSB_Init (void)

AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.

Hence you should just run AIOUSB_Init() once at the beginning and then the AIOUSB_Exit() once at the end after every thread acquiring data has been stopped.

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

34.117.3 Variable Documentation

AIOUSBDevice deviceTable[MAX_USB_DEVICES]

unsigned long AIOUSB_INIT_PATTERN = 0x9b6773adul

unsigned long aiousbInit = 0

34.118 lib/ndkbuild/jni/AIODeviceTable.c File Reference

```
#include "AIODeviceTable.h"
#include <string.h>
#include <errno.h>
```

Macros

- #define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

Functions

- AIOUSB_BOOL AIOUSB_SetInit ()
- void AIODeviceTableInit (void)
- AIOUSB_BOOL AIOUSB_IsInit ()
- unsigned long AIOUSB_InitTest (void)
- AIOUSB_BOOL AIOUSB_Cleanup ()
- unsigned long QueryDeviceInfo (unsigned long DeviceIndex, unsigned long *pPID, unsigned long *pNameSize, char *pName, unsigned long *pDIOBytes, unsigned long *pCounters)

  Identifying devices on the USB bus.
- PRIVATE char * ProductIDToName (unsigned int productID)

  this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()
- PRIVATE AIORET_TYPE ProductNameToID (const char *name)
- AIORET_TYPE GetDevices (void)
- USBDevice * AIODeviceTableGetUSBDeviceAtIndex (unsigned long DeviceIndex, AIORESULT *res)
- char * GetSafeDeviceName (unsigned long DeviceIndex)

  GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.
- AIORET_TYPE AIOUSB_EnsureOpen (unsigned long DeviceIndex)
- AIOUSBDevice * AIODeviceTableGetDeviceAtIndex (unsigned long DeviceIndex, AIORESULT *res)
- AIOUSBDevice * AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSBGetError ()
- AIORESULT AIODeviceTableAddDeviceToDeviceTable (int *numAccesDevices, unsigned long productID)

  A mock function that can set up the DeviceTable with any type of devices.
- AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int *numAccesDevices, unsigned long productID, USBDevice *usb_dev)
- AIORET_TYPE ClearAIODeviceTable (int numDevices)

  cleans up the AIODeviceTable and frees any memory associated with it.
- AIORESULT AIODeviceTableSetDeviceID (int index, AIOUSBDevice *dev)

- AIORESULT AIOUSB_GetAllDevices ()
- AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long *products, int length)
- void CloseAllDevices (void)
- unsigned long AIODeviceTableClearDevices (void)
- unsigned long ClearDevices (void)
- AIORET_TYPE AIODeviceTablePopulateTable (void)

  populate device table with ACCES devices found on USB bus
- AIORET_TYPE AIOUSB_Init (void)

  AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.
- AIORET_TYPE AIOUSB_Exit ()
- AIORET_TYPE AIOUSB_Reset (unsigned long DeviceIndex)

Variables

- AIOUSBDevice deviceTable [MAX_USB_DEVICES]
- unsigned long AIOUSB_INIT_PATTERN = 0x9b6773adul
- unsigned long aiousbInit = 0

34.118.1 Macro Definition Documentation

#define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

34.118.2 Function Documentation

AIOUSB_BOOL AIOUSB_SetInit ( )

void AIODeviceTableInit (void)

AIOUSB_BOOL AIOUSB_IsInit ( )

unsigned long AIOUSB_InitTest (void)

AIOUSB_BOOL AIOUSB_Cleanup ( )

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex*, unsigned long * *pPID*, unsigned long * *pNameSize*, char * *pName*, unsigned long * *pDIOBytes*, unsigned long * *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

PRIVATE char* ProductIDToName (unsigned int *productID*)

this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()

productIDIndex[] represents an index into productIDNameTable[], sorted by product ID; specifically, it contains pointers into productIDNameTable[]; to get the actual product ID, the pointer in productIDIndex[] must be dereferenced; using a separate index table instead of sorting productIDNameTable[] directly permits us to create multiple indexes, in particular, a second index sorted by product name

PRIVATE AIORET_TYPE ProductNameToID (const char * *name*)

This function is the complement of ProductIDToName() and returns the product ID for a given name; this function should be used with care; it will work reliably if passed a name obtained from ProductIDToName(); however, if passed a name obtained from the device itself it may not work; the reason is that devices contain their own name strings, which are most likely identical to the names defined in this module, but not guaranteed to be so; that's not as big a problem as it sounds, however, because if one has the means to obtain the name from the device, then they also have access to the device's product ID, so calling this function is unnecessary; this function is mainly for performing simple conversions between product names and IDs, primarily to support user interfaces

Parameters

name	

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if < 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid

USBDevice* AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice * A Usb handle that can be used for USB transactions

char* GetSafeDeviceName (unsigned long *DeviceIndex*)

GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.

AIORET_TYPE AIOUSB_EnsureOpen (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSBDevice* **AIODeviceTableGetDeviceAtIndex** (unsigned long *DeviceIndex,* **AIORESULT** * *res*)

AIOUSBDevice* **AIODeviceTableGetAIOUSBDeviceAtIndex** (unsigned long *DeviceIndex*)

AIORET_TYPE **AIOUSBGetError** ( )

AIORESULT **AIODeviceTableAddDeviceToDeviceTable** (int * *numAccesDevices,* unsigned long *productID*)

A mock function that can set up the DeviceTable with any type of devices.

AIORESULT **AIODeviceTableAddDeviceToDeviceTableWithUSBDevice** (int * *numAccesDevices,* unsigned long *productID,* **USBDevice** * *usb_dev*)

AIORET_TYPE **ClearAIODeviceTable** (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

AIORESULT **AIODeviceTableSetDeviceID** (int *index,* **AIOUSBDevice** * *dev*)

AIORESULT **AIOUSB_GetAllDevices** ( )

AIORET_TYPE **AIODeviceTablePopulateTableTest** (unsigned long * *products,* int *length*)

void **CloseAllDevices** (**void**)

unsigned long **AIODeviceTableClearDevices** (**void**)

unsigned long **ClearDevices** (**void**)

AIORET_TYPE **AIODeviceTablePopulateTable** (**void**)

populate device table with ACCES devices found on USB bus

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

Note

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

AIORET_TYPE **AIOUSB_Init** (**void**)

AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.

Hence you should just run AIOUSB_Init() once at the beginning and then the AIOUSB_Exit() once at the end after every thread acquiring data has been stopped.

AIORET_TYPE **AIOUSB_Exit** ( )

AIORET_TYPE **AIOUSB_Reset** (unsigned long *DeviceIndex*)

34.118.3 Variable Documentation

AIOUSBDevice **deviceTable[MAX_USB_DEVICES]**

unsigned long **AIOUSB_INIT_PATTERN** = 0x9b6773adul

unsigned long **aiousbInit** = 0

34.119 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

34.119.1 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;'

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

Note

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

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long * *products,* int *length*)

AIORESULT AIODeviceTableClearDevices (void)

AIORESULT ClearDevices (void)

AIOUSBDevice* AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

AIOUSBDevice* AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

USBDevice* AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice * A Usb handle that can be used for USB transactions

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex,* unsigned long * *pPID,* unsigned long * *pNameSize,* char * *pName,* unsigned long * *pDIOBytes,* unsigned long * *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if $<$ 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid
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

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

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

AIOUSB_BOOL AIOUSB_IsInit ( )

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

void AIODeviceTableInit (void)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

void CloseAllDevices (void)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIOUSBGetError ( )

34.119.2 Variable Documentation

AIOUSBDevice deviceTable[MAX_USB_DEVICES]

unsigned long AIOUSB_INIT_PATTERN

34.120 lib/ndkbuild/jni/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

34.120.1 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;'

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

Note

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

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long * *products,* int *length*)

AIORESULT AIODeviceTableClearDevices (void)

AIORESULT ClearDevices (void)

AIOUSBDevice* AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

AIOUSBDevice* AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

USBDevice* AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice * A Usb handle that can be used for USB transactions

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex*, unsigned long * *pPID*, unsigned long * *pNameSize*, char * *pName*, unsigned long * *pDIOBytes*, unsigned long * *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if < 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid
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

productIDIndex[] represents an index into productIDNameTable[], sorted by product ID; specifically, it contains pointers into productIDNameTable[]; to get the actual product ID, the pointer in productIDIndex[] must be dereferenced; using a separate index table instead of sorting productIDNameTable[] directly permits us to create multiple indexes, in particular, a second index sorted by product name

AIORET_TYPE ProductNameToID (const char * *name*)

This function is the complement of ProductIDToName() and returns the product ID for a given name; this function should be used with care; it will work reliably if passed a name obtained from ProductIDToName(); however, if passed a name obtained from the device itself it may not work; the reason is that devices contain their own name strings, which are most likely identical to the names defined in this module, but not guaranteed to be so; that's not as big a problem as it sounds, however, because if one has the means to obtain the name from the device, then they also have access to the device's product ID, so calling this function is unnecessary; this function is mainly for performing simple conversions between product names and IDs, primarily to support user interfaces

Parameters

name	

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

AIORET_TYPE AIOUSB_Init (void)

AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.

Hence you should just run AIOUSB_Init() once at the beginning and then the AIOUSB_Exit() once at the end after every thread acquiring data has been stopped.

AIORET_TYPE AIOUSB_EnsureOpen (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL AIOUSB_IsInit ( )

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

void AIODeviceTableInit (void)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

void CloseAllDevices (void)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIOUSBGetError ( )

34.120.2 Variable Documentation

AIOUSBDevice deviceTable[MAX_USB_DEVICES]

unsigned long AIOUSB_INIT_PATTERN

34.121 lib/AIOEither.c File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <assert.h>
#include <stdarg.h>
#include <stdio.h>
```

Macros

- #define LOOKUP(T) aioret_value_ ## T
- #define AIO_EITHER_CHECK_VALUE(RETVAL, TYPE)
- #define AIO_EITHER_GET_VALUE(RETVAL, TYPE)

Functions

- AIORET_TYPE AIOEitherClear (AIOEither *retval)
- AIORET_TYPE AIOEitherSetRight (AIOEither *retval, AIO_EITHER_TYPE val, void *tmp,...)
- AIORET_TYPE AIOEitherGetRight (AIOEither *retval, void *tmp,...)
- AIORET_TYPE AIOEitherSetLeft (AIOEither *retval, int val)
- AIORET_TYPE AIOEitherGetLeft (AIOEither *retval)
- AIOUSB_BOOL AIOEitherHasError (AIOEither *retval)
- char * AIOEitherToString (AIOEither *retval, AIORET_TYPE *result)
- int AIOEitherToInt (AIOEither retval)
- short AIOEitherToShort (AIOEither *retval, AIORET_TYPE *result)
- unsigned AIOEitherToUnsigned (AIOEither *retval, AIORET_TYPE *result)
- double AIOEitherToDouble (AIOEither *retval, AIORET_TYPE *result)
- AIO_NUMBER AIOEitherToAIONumber (AIOEither *retval, AIORET_TYPE *result)
- AIORET_TYPE AIOEitherToAIORetType (AIOEither either)

34.121.1 Macro Definition Documentation

#define LOOKUP(*T*) aioret_value_ ## T

#define AIO_EITHER_CHECK_VALUE(*RETVAL,* TYPE)

Value:

```
if ( RETVAL->left ) {                                   \
        *result = RETVAL->left;                             \
    } else {                                                \
        *result = AIOUSB_SUCCESS;                           \
        return *(TYPE *)&(RETVAL->right.number);            \
    }                                                       \
    return (TYPE)AIO_ERROR_VALUE;
```

#define AIO_EITHER_GET_VALUE(*RETVAL,* TYPE)

Value:

```
({                  \
            int tmp;                                            \
            if(RETVAL.left) {                                   \
                errno=RETVAL.left;                              \
                tmp=(TYPE)AIO_ERROR_VALUE;                      \
            } else {                                            \
                tmp=*(TYPE *)&(RETVAL.right.number);            \
            };                                                  \
            tmp;})
```

34.121.2 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*)

34.122 lib/ndkbuild/jni/AIOEither.c File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <assert.h>
#include <stdarg.h>
#include <stdio.h>
```

Macros

- #define LOOKUP(T) aioret_value_ ## T
- #define AIO_EITHER_CHECK_VALUE(RETVAL, TYPE)
- #define AIO_EITHER_GET_VALUE(RETVAL, TYPE)

Functions

- AIORET_TYPE AIOEitherClear (AIOEither *retval)
- AIORET_TYPE AIOEitherSetRight (AIOEither *retval, AIO_EITHER_TYPE val, void *tmp,...)
- AIORET_TYPE AIOEitherGetRight (AIOEither *retval, void *tmp,...)
- AIORET_TYPE AIOEitherSetLeft (AIOEither *retval, int val)
- AIORET_TYPE AIOEitherGetLeft (AIOEither *retval)
- AIOUSB_BOOL AIOEitherHasError (AIOEither *retval)
- char * AIOEitherToString (AIOEither *retval, AIORET_TYPE *result)
- int AIOEitherToInt (AIOEither retval)
- short AIOEitherToShort (AIOEither *retval, AIORET_TYPE *result)
- unsigned AIOEitherToUnsigned (AIOEither *retval, AIORET_TYPE *result)
- double AIOEitherToDouble (AIOEither *retval, AIORET_TYPE *result)
- AIO_NUMBER AIOEitherToAIONumber (AIOEither *retval, AIORET_TYPE *result)
- AIORET_TYPE AIOEitherToAIORetType (AIOEither either)

34.122.1 Macro Definition Documentation

#define LOOKUP(*T*) aioret_value_ ## T

#define AIO_EITHER_CHECK_VALUE(*RETVAL,* TYPE)

Value:

```
if ( RETVAL->left ) {                                   \
        *result = RETVAL->left;                             \
    } else {                                                \
        *result = AIOUSB_SUCCESS;                           \
        return *(TYPE *)&(RETVAL->right.number);            \
    }                                                       \
    return (TYPE)AIO_ERROR_VALUE;
```

#define AIO_EITHER_GET_VALUE(*RETVAL,* TYPE)

Value:

```
({                  \
            int tmp;                                        \
            if(RETVAL.left) {                               \
                errno=RETVAL.left;                          \
                tmp=(TYPE)AIO_ERROR_VALUE;                  \
            } else {                                        \
                tmp=*(TYPE *)&(RETVAL.right.number);        \
            };                                              \
            tmp;})
```

34.122.2 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*)

34.123 lib/AIOEither.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Data Structures

- struct aio_either_val
- struct aio_ret_value

Macros

- #define AIO_ERROR_VALUE 0xffffffffffffffff

Typedefs

- typedef struct aio_either_val AIO_EITHER_VALUE_ITEM
- typedef struct aio_ret_value AIOEither

Enumerations

- enum AIO_EITHER_TYPE {
  aioeither_value_int = 1, aioeither_value_int32_t = 1, aioeither_value_uint32_t = 2, aioeither_value_unsigned = 2, aioeither_value_uint16_t = 3, aioeither_vlaue_int16_t = 4, aioeither_value_double_t = 5, aioeither_value_double = 5,
  aioeither_value_uint8_t, aioeither_value_string, aioeither_value_longdouble_t, aioeither_value_obj,
  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)

34.123.1 Macro Definition Documentation

#define AIO_ERROR_VALUE 0xffffffffffffffff

34.123.2 Typedef Documentation

typedef struct aio_either_val AIO_EITHER_VALUE_ITEM

typedef struct aio_ret_value AIOEither

34.123.3 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

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

34.123.4 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*)

34.124 lib/ndkbuild/jni/AIOEither.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Data Structures

- struct aio_either_val
- struct aio_ret_value

Macros

- #define AIO_ERROR_VALUE 0xffffffffffffffff

Typedefs

- typedef struct aio_either_val AIO_EITHER_VALUE_ITEM
- typedef struct aio_ret_value AIOEither

Enumerations

- enum AIO_EITHER_TYPE {
  aioeither_value_int = 1, aioeither_value_int32_t = 1, aioeither_value_uint32_t = 2, aioeither_value_unsigned = 2, aioeither_value_uint16_t = 3, aioeither_vlaue_int16_t = 4, aioeither_value_double_t = 5, aioeither_value_double = 5,
  aioeither_value_uint8_t, aioeither_value_string, aioeither_value_longdouble_t, aioeither_value_obj,
  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)

34.124.1 Macro Definition Documentation

#define AIO_ERROR_VALUE 0xffffffffffffffff

34.124.2 Typedef Documentation

typedef struct aio_either_val AIO_EITHER_VALUE_ITEM

typedef struct aio_ret_value AIOEither

34.124.3 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

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

34.124.4 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*)

34.125 lib/AIOFifo.c File Reference

```
#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)

34.125.1 Macro Definition Documentation

#define LOOKUP(*T*) aioeither_value_ ## T

34.125.2 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)

34.126 lib/ndkbuild/jni/AIOFifo.c File Reference

```
#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)

34.126.1 Macro Definition Documentation

#define LOOKUP(*T*) aioeither_value_ ## T

34.126.2 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)

34.127 lib/AIOFifo.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
```

Data Structures

- struct AIOFifo

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

Macros

- #define LOCKING_MECHANISM ;
- #define GRAB_RESOURCE(obj) ;
- #define RELEASE_RESOURCE(obj) ;
- #define AIO_FIFO_INTERFACE
- #define TEMPLATE_AIOFIFO_INTERFACE(NAME, TYPE)
- #define TEMPLATE_AIOFIFO_API(NAME, TYPE)

Typedefs

- typedef struct AIOFifo AIOFifo

  AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.
- typedef uint32_t TYPE
- typedef void INPUT_TYPE
- typedef struct new_aio_fifo AIOFifoTYPE

Functions

- TEMPLATE_AIOFIFO_INTERFACE (Counts, uint16_t)

  Counts Fifo definition that is a Fifo of 2 byte count values.
- TEMPLATE_AIOFIFO_INTERFACE (Volts, double)

  Volts Fifo definition that is a Fifo of 8 byte double values that will be analog voltage readings.
- AIOFifo * NewAIOFifo (unsigned int size, unsigned int refsize)
- void DeleteAIOFifo (AIOFifo *fifo)
- void AIOFifoReset (void *fifo)
- AIORET_TYPE AIOFifoRead (AIOFifo *fifo, void *tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWrite (AIOFifo *fifo, void *frombuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo *fifo, void *frombuf, unsigned maxsize)

  for AllOrNoneTesting
- AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo *fifo, void *tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoGetRefSize (void *fifo)
- AIOFifoTYPE * NewAIOFifoTYPE (unsigned int size)
- AIORET_TYPE Push (AIOFifoTYPE *fifo, TYPE a)
- AIORET_TYPE PushN (AIOFifoTYPE *fifo, INPUT_TYPE *a, unsigned N)
- AIORET_TYPE PopN (AIOFifoTYPE *fifo, INPUT_TYPE *a, unsigned N)
- AIORET_TYPE AIOFifoWriteSizeRemaining (void *fifo)
- AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void *fifo)
- AIORET_TYPE AIOFifoReadSize (void *tmpfifo)
- AIORET_TYPE AIOFifoReadSizeNumElements (void *tmpfifo)
- AIORET_TYPE AIOFifoGetSize (void *fifo)
- AIORET_TYPE AIOFifoGetSizeNumElements (void *tmpfifo)
- AIORET_TYPE AIOFifoResize (AIOFifo *fifo, size_t newsize)
- AIORET_TYPE AIOFifoReadPosition (void *nfifo)
- AIORET_TYPE AIOFifoWritePosition (void *nfifo)

34.127.1 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)

34.127.2 Typedef Documentation

typedef struct **AIOFifo AIOFifo**

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

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

typedef **uint32_t TYPE**

typedef **void INPUT_TYPE**

typedef struct **new_aio_fifo AIOFifoTYPE**

34.127.3 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*)

34.128 lib/ndkbuild/jni/AIOFifo.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
```

Data Structures

- struct AIOFifo

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

Macros

- #define LOCKING_MECHANISM ;

- #define GRAB_RESOURCE(obj) ;
- #define RELEASE_RESOURCE(obj) ;
- #define AIO_FIFO_INTERFACE
- #define TEMPLATE_AIOFIFO_INTERFACE(NAME, TYPE)
- #define TEMPLATE_AIOFIFO_API(NAME, TYPE)

Typedefs

- typedef struct AIOFifo AIOFifo

  AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.
- typedef uint32_t TYPE
- typedef void INPUT_TYPE
- typedef struct new_aio_fifo AIOFifoTYPE

Functions

- TEMPLATE_AIOFIFO_INTERFACE (Counts, uint16_t)

  Counts Fifo definition that is a Fifo of 2 byte count values.
- TEMPLATE_AIOFIFO_INTERFACE (Volts, double)

  Volts Fifo definition that is a Fifo of 8 byte double values that will be analog voltage readings.
- AIOFifo * NewAIOFifo (unsigned int size, unsigned int refsize)
- void DeleteAIOFifo (AIOFifo *fifo)
- void AIOFifoReset (void *fifo)
- AIORET_TYPE AIOFifoRead (AIOFifo *fifo, void *tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWrite (AIOFifo *fifo, void *frombuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo *fifo, void *frombuf, unsigned maxsize)

  for AllOrNoneTesting
- AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo *fifo, void *tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoGetRefSize (void *fifo)
- AIOFifoTYPE * NewAIOFifoTYPE (unsigned int size)
- AIORET_TYPE Push (AIOFifoTYPE *fifo, TYPE a)
- AIORET_TYPE PushN (AIOFifoTYPE *fifo, INPUT_TYPE *a, unsigned N)
- AIORET_TYPE PopN (AIOFifoTYPE *fifo, INPUT_TYPE *a, unsigned N)
- AIORET_TYPE AIOFifoWriteSizeRemaining (void *fifo)
- AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void *fifo)
- AIORET_TYPE AIOFifoReadSize (void *tmpfifo)
- AIORET_TYPE AIOFifoReadSizeNumElements (void *tmpfifo)
- AIORET_TYPE AIOFifoGetSize (void *fifo)
- AIORET_TYPE AIOFifoGetSizeNumElements (void *tmpfifo)
- AIORET_TYPE AIOFifoResize (AIOFifo *fifo, size_t newsize)
- AIORET_TYPE AIOFifoReadPosition (void *nfifo)
- AIORET_TYPE AIOFifoWritePosition (void *nfifo)

34.128.1 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)

34.128.2 Typedef Documentation

typedef struct **AIOFifo AIOFifo**

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

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

typedef **uint32_t TYPE**

typedef **void INPUT_TYPE**

typedef struct **new_aio_fifo AIOFifoTYPE**

34.128.3 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*)

34.129 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

34.129.1 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*)

34.129.2 Variable Documentation

int newone

34.130 lib/ndkbuild/jni/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

34.130.1 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*)

34.130.2 Variable Documentation

int newone

34.131 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)

34.131.1 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)

34.131.2 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*)

34.132 lib/ndkbuild/jni/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)

34.132.1 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)

34.132.2 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*)

34.133 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)

34.133.1 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*)

34.134 lib/ndkbuild/jni/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_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_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)

34.134.1 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*)

34.135 lib/AIOProductTypes.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct AIOProductRange

  A simplified range of Products based off of device ids.
- struct AIOProductGroup

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

Macros

- #define NUMARGS(...) (sizeof((void∗[]){__VA_ARGS__})/sizeof(void∗))
- #define AIO_RANGE(start, stop) (&(AIOProductRange){ ._start=start, ._end =stop })
- #define AIO_PRODUCT_GROUP(NAME, N,...) const AIOProductGroup NAME = { ._num_groups =N, ._groups = (AIOProductRange ∗∗)&(AIOProductRange ∗[N]){ __VA_ARGS__ } } ;
- #define AIO_PRODUCT_CONSTANT(NAME, NAMEPTR, NAMEFN, N,...)
- #define AIO_PRODUCT_EXTERN(NAME, NAMEPTR, NAMEFN)

Typedefs

- typedef struct AIOProductRange AIOProductRange

  A simplified range of Products based off of device ids.
- typedef struct AIOProductGroup AIOProductGroup

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

Functions

- AIOProductRange ∗ NewAIOProductRange (unsigned long start, unsigned long end)
- AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗pr)
- AIOProductGroup ∗ NewAIOProductGroup (size_t numgroups,...)
- AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)
- AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗g, unsigned long val)
- AIOProductGroup ∗ groupcpy (const AIOProductGroup ∗g)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ, AIO_ANALOG_OUTPUT_GROUP, AIO_ANALOG_-OUTPUT)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ, AIO_ANALOG_INPUT_GROUP, AIO_ANALOG_INPU-T)

- AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ, AIO_DIGITAL_HIGHSPEED_GROUP, AIO_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)

34.135.1 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__); \
        AIOProductGroup *NAMEPTR = &NAME; \
        *NAMEFN() { return groupcpy( NAMEPTR );}
```

```
const
AIOProductGroup
```

#define AIO_PRODUCT_EXTERN(*NAME, NAMEPTR, NAMEFN*)

Value:

```
extern const AIOProductGroup NAME; \
      AIOProductGroup *NAMEPTR;\
```

```
extern const
extern AIOProductGroup *NAMEFN();
```

34.135.2 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.

34.135.3 Function Documentation

AIOProductRange∗ NewAIOProductRange (unsigned long *start,* unsigned long *end*)

AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗ *pr*)

AIOProductGroup∗ NewAIOProductGroup (size_t *numgroups, ...*)

AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)

AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗ *g,* unsigned long *val*)

AIOProductGroup∗ groupcpy (const AIOProductGroup ∗ *g*)

AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ , AIO_ANALOG_OUTPUT_GROUP , AIO_ANALOG_OUTPUT)

AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ , AIO_ANALOG_INPUT_GROUP , AIO_ANALOG_INPUT)

AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ , AIO_DIGITAL_HIGHSPEED_GROUP , AIO_DIGITAL_HIGHSPEED)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_100KHZ_OBJ , AIO_ANALOG_CLOCK_100KHZ_GROUP , AIO_ANALOG_CLOCK_100KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_250KHZ_OBJ , AIO_ANALOG_CLOCK_250KHZ_GROUP , AIO_ANALOG_CLOCK_250KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_500KHZ_OBJ , AIO_ANALOG_CLOCK_500KHZ_GROUP , AIO_ANALOG_CLOCK_500KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_12BIT_A2D_OBJ , AIO_ANALOG_12BIT_A2D_GROUP , AIO_AIO_ANALOG_12BIT_A2D)

AIO_PRODUCT_EXTERN (AIO_ANALOG_16BIT_A2D_OBJ , AIO_ANALOG_16BIT_A2D_GROUP , AIO_AIO_ANALOG_16BIT_A2D)

34.136 lib/ndkbuild/jni/AIOProductTypes.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct AIOProductRange

  A simplified range of Products based off of device ids.
- struct AIOProductGroup

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

Macros

- #define NUMARGS(...) (sizeof((void∗[]){__VA_ARGS__})/sizeof(void∗))
- #define AIO_RANGE(start, stop) (&(AIOProductRange){ ._start=start, ._end =stop })
- #define AIO_PRODUCT_GROUP(NAME, N,...) const AIOProductGroup NAME = { ._num_groups =N, ._groups = (AIOProductRange ∗∗)&(AIOProductRange ∗[N]){ __VA_ARGS__ } } ;
- #define AIO_PRODUCT_CONSTANT(NAME, NAMEPTR, NAMEFN, N,...)
- #define AIO_PRODUCT_EXTERN(NAME, NAMEPTR, NAMEFN)

Typedefs

- typedef struct AIOProductRange AIOProductRange

  A simplified range of Products based off of device ids.
- typedef struct AIOProductGroup AIOProductGroup

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

Functions

- AIOProductRange ∗ NewAIOProductRange (unsigned long start, unsigned long end)
- AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗pr)
- AIOProductGroup ∗ NewAIOProductGroup (size_t numgroups,...)
- AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)
- AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗g, unsigned long val)
- AIOProductGroup ∗ groupcpy (const AIOProductGroup ∗g)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ, AIO_ANALOG_OUTPUT_GROUP, AIO_ANALOG_-OUTPUT)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ, AIO_ANALOG_INPUT_GROUP, AIO_ANALOG_INPU-T)
- AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ, AIO_DIGITAL_HIGHSPEED_GROUP, AIO_DIG-ITAL_HIGHSPEED)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_100KHZ_OBJ, AIO_ANALOG_CLOCK_100KHZ_GROUP, AIO_ANALOG_CLOCK_100KHZ)

- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_250KHZ_OBJ, AIO_ANALOG_CLOCK_250KHZ_GROUP, AIO_ANALOG_CLOCK_250KHZ)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_500KHZ_OBJ, AIO_ANALOG_CLOCK_500KHZ_GROUP, AIO_ANALOG_CLOCK_500KHZ)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_12BIT_A2D_OBJ, AIO_ANALOG_12BIT_A2D_GROUP, AIO_AIO_-ANALOG_12BIT_A2D)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_16BIT_A2D_OBJ, AIO_ANALOG_16BIT_A2D_GROUP, AIO_AIO_-ANALOG_16BIT_A2D)

34.136.1 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__); \
        AIOProductGroup *NAMEPTR = &NAME; \
        *NAMEFN() { return groupcpy( NAMEPTR );}
```

```
const
AIOProductGroup
```

#define AIO_PRODUCT_EXTERN(*NAME, NAMEPTR, NAMEFN*)

Value:

```
extern const AIOProductGroup NAME; \
       AIOProductGroup *NAMEPTR;\
```

```
extern const
extern AIOProductGroup *NAMEFN();
```

34.136.2 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.

34.136.3 Function Documentation

AIOProductRange∗ NewAIOProductRange (unsigned long *start,* unsigned long *end*)

AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗ *pr*)

AIOProductGroup∗ NewAIOProductGroup (size_t *numgroups, ...*)

AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)

AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗ *g,* unsigned long *val*)

AIOProductGroup∗ groupcpy (const AIOProductGroup ∗ *g*)

AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ , AIO_ANALOG_OUTPUT_GROUP , AIO_ANALOG_OUTPUT)

AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ , AIO_ANALOG_INPUT_GROUP , AIO_ANALOG_INPUT)

AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ , AIO_DIGITAL_HIGHSPEED_GROUP , AIO_DIGITAL_HIGHSPEED)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_100KHZ_OBJ , AIO_ANALOG_CLOCK_100KHZ_GROUP , AIO_ANALOG_CLOCK_100KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_250KHZ_OBJ , AIO_ANALOG_CLOCK_250KHZ_GROUP , AIO_ANALOG_CLOCK_250KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_500KHZ_OBJ , AIO_ANALOG_CLOCK_500KHZ_GROUP , AIO_ANALOG_CLOCK_500KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_12BIT_A2D_OBJ , AIO_ANALOG_12BIT_A2D_GROUP , AIO_AIO_ANALOG_12BIT_A2D)

AIO_PRODUCT_EXTERN (AIO_ANALOG_16BIT_A2D_OBJ , AIO_ANALOG_16BIT_A2D_GROUP , AIO_AIO_ANALOG_16BIT_A2D)

34.137 lib/AIOTuple.c File Reference

```
#include "AIOTuple.h"
#include <stdarg.h>
#include <string.h>
```

34.138 lib/ndkbuild/jni/AIOTuple.c File Reference

```
#include "AIOTuple.h"
#include <stdarg.h>
#include <string.h>
```

34.139 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)

34.139.1 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)

34.139.2 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]

34.140 lib/ndkbuild/jni/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)

34.140.1 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)

34.140.2 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]`

34.141 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_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,...)
- #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, 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, 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_NORMAL, AIOCONTINUOUS_BUF_OVERRIDE,
  AIOContinuousBufMode_end, AIOContinuousBufMode_begin = (0 -1), AIOCONTINUOUS_BUF_ALLORNONE, AIOCONTINUOUS_BUF_NORMAL,
  AIOCONTINUOUS_BUF_OVERRIDE, AIOContinuousBufMode_end }
- enum { MAX_USB_DEVICES = 32 }
- enum AIOUSB_BOOL_VAL { AIOUSB_FALSE = 0, AIOUSB_TRUE = 1, 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,
ProductIDS_begin = (0 -1), ACCES_VENDOR_ID = 0x1605, USB_DA12_8A_REV_A = 0xC001, USB_DA12_8A = 0xC002,
USB_DA12_8E = 0xC003, USB_DIO_32 = 0x8001, USB_DIO_32I = 0x8004, USB_DIO_48 = 0x8002,
USB_DIO_96 = 0x8003, USB_DIO24_CTR6 = 0x8006, USB_DI16A_REV_A1 = 0x8008, USB_DO16A_REV_A1 = 0x8009,
USB_DI16A_REV_A2 = 0x800a, USB_DIO_16H = 0x800c, USB_DI16A = 0x800d, USB_DO16A = 0x800e,
USB_DIO_16A = 0x800f, USB_IIRO_16 = 0x8010, USB_II_16 = 0x8011, USB_RO_16 = 0x8012,
USB_IIRO_8 = 0x8014, USB_II_8 = 0x8015, USB_IIRO_4 = 0x8016, USB_IDIO_16 = 0x8018,
USB_II_16_OLD = 0x8019, USB_IDO_16 = 0x801a, USB_IDIO_8 = 0x801c, USB_II_8_OLD = 0x801d,
USB_IDIO_4 = 0x801e, USB_CTR_15 = 0x8020, USB_IIRO4_2SM = 0x8030, USB_IIRO4_COM = 0x8031,
USB_DIO16RO8 = 0x8032, USB_DIO48DO24 = 0x803C, USB_DIO24DO12 = 0x803D, USB_DO24 = 0x803E,
PICO_DIO16RO8 = 0x8033, USBP_II8IDO4A = 0x8036, USB_AI16_16A = 0x8040, USB_AI16_16E = 0x8041,
USB_AI12_16A = 0x8042, USB_AI12_16 = 0x8043, USB_AI12_16E = 0x8044, USB_AI16_64MA = 0x8045,
USB_AI16_64ME = 0x8046, USB_AI12_64MA = 0x8047, USB_AI12_64M = 0x8048, USB_AI12_64ME = 0x8049,
USB_AI16_32A = 0x804a, USB_AI16_32E = 0x804b, USB_AI12_32A = 0x804c, USB_AI12_32 = 0x804d,
USB_AI12_32E = 0x804e, USB_AI16_64A = 0x804f, USB_AI16_64E = 0x8050, USB_AI12_64A = 0x8051,
USB_AI12_64 = 0x8052, USB_AI12_64E = 0x8053, USB_AI16_96A = 0x8054, USB_AI16_96E = 0x8055,
USB_AI12_96A = 0x8056, USB_AI12_96 = 0x8057, USB_AI12_96E = 0x8058, USB_AI16_128A = 0x8059,
USB_AI16_128E = 0x805a, USB_AI12_128A = 0x805b, USB_AI12_128 = 0x805c, USB_AI12_128E = 0x805d,
USB_AO_ARB1 = 0x8068, USB_AO16_16A = 0x8070, USB_AO16_16 = 0x8071, USB_AO16_12A = 0x8072,
USB_AO16_12 = 0x8073, USB_AO16_8A = 0x8074, USB_AO16_8 = 0x8075, USB_AO16_4A = 0x8076,
USB_AO16_4 = 0x8077, USB_AO12_16A = 0x8078, USB_AO12_16 = 0x8079, USB_AO12_12A = 0x807a,
USB_AO12_12 = 0x807b, USB_AO12_8A = 0x807c, USB_AO12_8 = 0x807d, USB_AO12_4A = 0x807e,
USB_AO12_4 = 0x807f, USB_AIO16_16A = 0x8140, USB_AIO16_16E = 0x8141, USB_AIO12_16A = 0x8142,
USB_AIO12_16 = 0x8143, USB_AIO12_16E = 0x8144, USB_AIO16_64MA = 0x8145, USB_AIO16_64ME = 0x8146,
USB_AIO12_64MA = 0x8147, USB_AIO12_64M = 0x8148, USB_AIO12_64ME = 0x8149, USB_AIO16_32A = 0x814a,
USB_AIO16_32E = 0x814b, USB_AIO12_32A = 0x814c, USB_AIO12_32 = 0x814d, USB_AIO12_32E = 0x814e,
USB_AIO16_64A = 0x814f, USB_AIO16_64E = 0x8150, USB_AIO12_64A = 0x8151, USB_AIO12_64 = 0x8152,
USB_AIO12_64E = 0x8153, USB_AIO16_96A = 0x8154, USB_AIO16_96E = 0x8155, USB_AIO12_96A = 0x8156,
USB_AIO12_96 = 0x8157, USB_AIO12_96E = 0x8158, USB_AIO16_128A = 0x8159, USB_AIO16_128E = 0x815a,
USB_AIO12_128A = 0x815b, USB_AIO12_128 = 0x815c, USB_AIO12_128E = 0x815d, ProductIDS_end }

- enum { diFirst = 0xFFFFFFFEul, diOnly = 0xFFFFFFFDul, diNone = 0xFFFFFFFFul }
- enum DACRange {
  DACRange_begin = (0 -1), DAC_RANGE_0_5V, DAC_RANGE_5V, DAC_RANGE_0_10V,
  DAC_RANGE_10V, DACRange_end, 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_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, 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, AIOU-

SB_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, 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, AD_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, 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_READLATCHED = 0x26, AUR_CTR_COS_BULK_GATE2 = 0x27, AUR_CTR_PUR_FIRST = 0x28, AUR_CTR_PUR_OFRQ = 0x28, AUR_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_STATE = 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_EEPROM_WRITE = 0XA2, AUR_DAC_CONTROL = 0xB0, AUR_DAC_DATAPTR = 0xB1, AUR_DAC_DIVISOR = 0xB2, AUR_DAC_IMMEDIATE = 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_START_ACQUIRING_BLOCK = 0xBC, AUR_DIO_STREAM_OPEN_INPUT = 0xBC,
AUR_DIO_SETCLOCKS = 0xBD, AUR_ADC_SET_CONFIG = 0xBE, AUR_ADC_IMMEDIATE = 0xBF, AUR_DIO_SPI_WRITE = 0xC0,
AUR_DIO_SPI_READ = 0xC1, AUR_ADC_GET_CONFIG = 0xD2, CYPRESS_GET_DESC = 0x06, VENDOR_REQUEST_end,
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_READLATCHED = 0x26,
AUR_CTR_COS_BULK_GATE2 = 0x27, AUR_CTR_PUR_FIRST = 0x28, AUR_CTR_PUR_OFRQ = 0x28, AUR_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_STATE = 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_EEPROM_WRITE = 0XA2,
AUR_DAC_CONTROL = 0xB0, AUR_DAC_DATAPTR = 0xB1, AUR_DAC_DIVISOR = 0xB2, AUR_DAC_IMMEDIATE = 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_START_ACQUIRING_BLOCK = 0xBC, AUR_DIO_STREAM_OPEN_INPUT = 0xBC,
AUR_DIO_SETCLOCKS = 0xBD, AUR_ADC_SET_CONFIG = 0xBE, AUR_ADC_IMMEDIATE = 0xBF, AUR_DIO_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_PARAMS = 0x0302,
  CYPRESS_MAX_DESC_SIZE = 256, AIOUSB_MAX_NAME_SIZE = 100, EEPROM_SERIAL_NUMBER_ADDRESS = 0x1DF8, EEPROM_CUSTOM_BASE_ADDRESS = 0x1E00,
  EEPROM_CUSTOM_MIN_ADDRESS = 0, EEPROM_CUSTOM_MAX_ADDRESS = 0x1FF, AD_CONFIG_REGISTERS = 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_REFERENCE = 3,
  AD_CAL_MODE_BIP_GROUND = 5, AD_CAL_MODE_HIGH_REF = 7, ADCalMode_end = 8, ADCalMode_begin = -1,
  AD_CAL_MODE_NORMAL = 0, AD_CAL_MODE_GROUND = 1, AD_CAL_MODE_REFERENCE = 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_RUNNING, AIO_TERMINATE_CALLBACK,
  AIOCommandCode_end, AIOCommandCode_begin = (0 -1), GENERIC_DOSOMETHING_PLACEHOLDER, AIO_CONTINUE_RUNNING,
  AIO_TERMINATE_CALLBACK, AIOCommandCode_end }

  Enums that govern how commands are performed and operated.

34.141.1 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_AIOCHANNEL-MASK, __VA_ARGS__)

#define AIO_ASSERT_CONFIG(*...*) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_ADCCONFIG , __VA_ARGS__)

#define AIO_ASSERT_AIOCONTBUF(*...*) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_AIOCONTINUOU-S_BUFFER, __VA_ARGS__);

#define AIO_ASSERT_AIOEITHER(*err, msg, ...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { AIOEither tmp; tmp.left = err; tmp.errmsg=strdup(msg); return tmp; }

#define AIO_ERROR(*X*) (-abs(X))

AIO_ERROR ∗ are just like the regular ASSERTIONS meaning that they argument should evaluate to true otherwise it will fail.

#define AIO_ERROR_VALID_DATA(*err, ...*) if (!(__VA_ARGS__)) { return err; }

#define AIO_ERROR_VALID_DATA_RETVAL(*err, ...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_VALID_AIORET_TYPE(*err, ...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_AIOEITHER_VALID_DATA(*err, ...*)

Value:

```
if ( !(__VA_ARGS__) ) { \
        AIOEither tmp; tmp.left = err; tmp.errmsg=NULL; return tmp; }
```

#define AIO_ERROR_VALID_DATA_W_CODE(*err, code, ...*) if (!(__VA_ARGS__)) { { code; }; return err; }

#define AIO_ERROR_VALID_DATA_WITH_CODE(*retval, err, code*) if (! (code)) { errno = -abs(err); return retval; }

#define G_STMT_START do

#define G_STMT_END while (0)

#define G_BREAKPOINT( ) G_STMT_START{ raise (SIGTRAP); }G_STMT_END

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                             \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;                 \
            goto g;                                             \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {            \
            goto g;                                             \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {    \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;                \
            goto g;                                             \
        }                                                       \
    } while (0 )
```

#define AIOUSB_ERROR_OFFSET 100

#define LIBUSB_RESULT_TO_AIOUSB_RESULT(*code*) (unsigned long)(AIOUSB_ERROR_OFFSET + -(int)(code))

#define AIOUSB_RESULT_TO_LIBUSB_RESULT(*code*) (-((int)(code) - AIOUSB_ERROR_OFFSET))

#define ROOTCLOCK 10000000

34.141.2 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**

34.141.3 Enumeration Type Documentation

enum **AIO_SCAN_TYPE**

Enumerator

AIO_PER_OVERSAMPLE
AIO_PER_CHANNEL
AIO_PER_SCANS
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
THREAD_STATUS_begin
INVALID_OBJECT
NOT_STARTED
RUNNING
WITH_DATA
TERMINATED
RUNNING_OR_WITH_DATA
JOINED
TERMINATED_OVERRUN
TERMINATING
THREAD_STATUS_end

enum AIOContinuousBufMode

Enumerator

AIOContinuousBufMode_begin

AIOCONTINUOUS_BUF_ALLORNONE

AIOCONTINUOUS_BUF_NORMAL

AIOCONTINUOUS_BUF_OVERRIDE

AIOContinuousBufMode_end

AIOContinuousBufMode_begin

AIOCONTINUOUS_BUF_ALLORNONE

AIOCONTINUOUS_BUF_NORMAL

AIOCONTINUOUS_BUF_OVERRIDE

AIOContinuousBufMode_end

anonymous enum

Enumerator

MAX_USB_DEVICES

enum AIOUSB_BOOL_VAL

other libraries often declare BOOL, TRUE and FALSE, and worse, they declare these using #define; so we sidestep that potential conflict by declaring the same types prefixed with AIOUSB_; it's ugly, but if people want to use the shorter names and they are certain they won't conflict with anything else, they can define the ENABLE_BOOL_TYPE macro

Enumerator

AIOUSB_FALSE

AIOUSB_TRUE

AIOUSB_FALSE

AIOUSB_TRUE

enum ProductIDS

Enumerator

ProductIDS_begin

ACCES_VENDOR_ID

USB_DA12_8A_REV_A • these product IDs are constant ∗

USB_DA12_8A

USB_DA12_8E

USB_DIO_32 • these are the product IDs after firmware is uploaded to the device; prior ∗ to uploading the firmware, the product ID is that shown below minus 0x8000

USB_DIO_32I

USB_DIO_48

USB_DIO_96

USB_DIO24_CTR6

USB_DI16A_REV_A1

USB_DO16A_REV_A1

USB_DI16A_REV_A2

USB_DIO_16H

USB_DI16A

USB_DO16A

USB_DIO_16A

USB_IIRO_16

USB_II_16

USB_RO_16

USB_IIRO_8

USB_II_8

USB_IIRO_4
USB_IDIO_16
USB_II_16_OLD
USB_IDO_16
USB_IDIO_8
USB_II_8_OLD
USB_IDIO_4
USB_CTR_15
USB_IIRO4_2SM
USB_IIRO4_COM
USB_DIO16RO8
USB_DIO48DO24
USB_DIO24DO12
USB_DO24
PICO_DIO16RO8
USBP_II8IDO4A
USB_AI16_16A
USB_AI16_16E
USB_AI12_16A
USB_AI12_16
USB_AI12_16E
USB_AI16_64MA
USB_AI16_64ME
USB_AI12_64MA
USB_AI12_64M
USB_AI12_64ME
USB_AI16_32A
USB_AI16_32E
USB_AI12_32A
USB_AI12_32
USB_AI12_32E
USB_AI16_64A
USB_AI16_64E
USB_AI12_64A
USB_AI12_64
USB_AI12_64E
USB_AI16_96A
USB_AI16_96E
USB_AI12_96A
USB_AI12_96
USB_AI12_96E
USB_AI16_128A
USB_AI16_128E
USB_AI12_128A
USB_AI12_128
USB_AI12_128E
USB_AO_ARB1
USB_AO16_16A
USB_AO16_16
USB_AO16_12A
USB_AO16_12
USB_AO16_8A
USB_AO16_8
USB_AO16_4A
USB_AO16_4

USB_AO12_16A

USB_AO12_16

USB_AO12_12A

USB_AO12_12

USB_AO12_8A

USB_AO12_8

USB_AO12_4A

USB_AO12_4

USB_AIO16_16A

USB_AIO16_16E

USB_AIO12_16A

USB_AIO12_16

USB_AIO12_16E

USB_AIO16_64MA

USB_AIO16_64ME

USB_AIO12_64MA

USB_AIO12_64M

USB_AIO12_64ME

USB_AIO16_32A

USB_AIO16_32E

USB_AIO12_32A

USB_AIO12_32

USB_AIO12_32E

USB_AIO16_64A

USB_AIO16_64E

USB_AIO12_64A

USB_AIO12_64

USB_AIO12_64E

USB_AIO16_96A

USB_AIO16_96E

USB_AIO12_96A

USB_AIO12_96

USB_AIO12_96E

USB_AIO16_128A

USB_AIO16_128E

USB_AIO12_128A

USB_AIO12_128

USB_AIO12_128E

ProductIDS_end

ProductIDS_begin

ACCES_VENDOR_ID

USB_DA12_8A_REV_A • these product IDs are constant ∗

USB_DA12_8A

USB_DA12_8E

USB_DIO_32 • these are the product IDs after firmware is uploaded to the device; prior ∗ to uploading the firmware, the product ID is that shown below minus 0x8000

USB_DIO_32I

USB_DIO_48

USB_DIO_96

USB_DIO24_CTR6

USB_DI16A_REV_A1

USB_DO16A_REV_A1

USB_DI16A_REV_A2

USB_DIO_16H

USB_DI16A

USB_DO16A

USB_DIO_16A
USB_IIRO_16
USB_II_16
USB_RO_16
USB_IIRO_8
USB_II_8
USB_IIRO_4
USB_IDIO_16
USB_II_16_OLD
USB_IDO_16
USB_IDIO_8
USB_II_8_OLD
USB_IDIO_4
USB_CTR_15
USB_IIRO4_2SM
USB_IIRO4_COM
USB_DIO16RO8
USB_DIO48DO24
USB_DIO24DO12
USB_DO24
PICO_DIO16RO8
USBP_II8IDO4A
USB_AI16_16A
USB_AI16_16E
USB_AI12_16A
USB_AI12_16
USB_AI12_16E
USB_AI16_64MA
USB_AI16_64ME
USB_AI12_64MA
USB_AI12_64M
USB_AI12_64ME
USB_AI16_32A
USB_AI16_32E
USB_AI12_32A
USB_AI12_32
USB_AI12_32E
USB_AI16_64A
USB_AI16_64E
USB_AI12_64A
USB_AI12_64
USB_AI12_64E
USB_AI16_96A
USB_AI16_96E
USB_AI12_96A
USB_AI12_96
USB_AI12_96E
USB_AI16_128A
USB_AI16_128E
USB_AI12_128A
USB_AI12_128
USB_AI12_128E
USB_AO_ARB1
USB_AO16_16A
USB_AO16_16

USB_AO16_12A

USB_AO16_12

USB_AO16_8A

USB_AO16_8

USB_AO16_4A

USB_AO16_4

USB_AO12_16A

USB_AO12_16

USB_AO12_12A

USB_AO12_12

USB_AO12_8A

USB_AO12_8

USB_AO12_4A

USB_AO12_4

USB_AIO16_16A

USB_AIO16_16E

USB_AIO12_16A

USB_AIO12_16

USB_AIO12_16E

USB_AIO16_64MA

USB_AIO16_64ME

USB_AIO12_64MA

USB_AIO12_64M

USB_AIO12_64ME

USB_AIO16_32A

USB_AIO16_32E

USB_AIO12_32A

USB_AIO12_32

USB_AIO12_32E

USB_AIO16_64A

USB_AIO16_64E

USB_AIO12_64A

USB_AIO12_64

USB_AIO12_64E

USB_AIO16_96A

USB_AIO16_96E

USB_AIO12_96A

USB_AIO12_96

USB_AIO12_96E

USB_AIO16_128A

USB_AIO16_128E

USB_AIO12_128A

USB_AIO12_128

USB_AIO12_128E

ProductIDS_end

anonymous enum

Enumerator

diFirst

diOnly

diNone

enum DACRange

range codes passed to DACSetBoardRange()

Enumerator

DACRange_begin
DAC_RANGE_0_5V
DAC_RANGE_5V
DAC_RANGE_0_10V
DAC_RANGE_10V
DACRange_end
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
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
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
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

ADGainCode_begin

AD_GAIN_CODE_0_10V

AD_GAIN_CODE_10V

AD_GAIN_CODE_0_5V

AD_GAIN_CODE_5V

AD_GAIN_CODE_0_2V

AD_GAIN_CODE_2V

AD_GAIN_CODE_0_1V

AD_GAIN_CODE_1V

ADGainCode_end

enum **VENDOR_REQUEST**

Enumerator

VENDOR_REQUEST_begin
AUR_DIO_WRITE
AUR_DIO_READ
AUR_DIO_CONFIG
AUR_DIO_CONFIG_QUERY
AUR_CTR_READ
AUR_CTR_MODE
AUR_CTR_LOAD
AUR_CTR_MODELOAD
AUR_CTR_SELGATE
AUR_CTR_READALL
AUR_CTR_READLATCHED
AUR_CTR_COS_BULK_GATE2
AUR_CTR_PUR_FIRST
AUR_CTR_PUR_OFRQ
AUR_CTR_COS_BULK_ABORT
AUR_CTR_PUR_MFRQ
AUR_CTR_PUR_EVCT
AUR_CTR_PUR_MPUL
AUR_WDG_STATUS
AUR_DIO_WDG16_DEPREC
AUR_READBACK_GLOBAL_STATE
AUR_SAVE_GLOBAL_STATE
AUR_GEN_CLEAR_FIFO_NEXT
AUR_GEN_CLEAR_FIFO
AUR_GEN_CLEAR_FIFO_WAIT
AUR_GEN_ABORT_AND_CLEAR
AUR_WDG
AUR_OFFLINE_READWRITE
AUR_SELF_TEST_1
AUR_EEPROM_READ
AUR_EEPROM_WRITE
AUR_DAC_CONTROL
AUR_DAC_DATAPTR
AUR_DAC_DIVISOR
AUR_DAC_IMMEDIATE
AUR_GEN_STREAM_STATUS
AUR_FLASH_READWRITE
AUR_DAC_RANGE
AUR_PROBE_CALFEATURE
AUR_LOAD_BULK_CALIBRATION_BLOCK
AUR_DIO_STREAM_OPEN_OUTPUT
AUR_START_ACQUIRING_BLOCK
AUR_DIO_STREAM_OPEN_INPUT
AUR_DIO_SETCLOCKS
AUR_ADC_SET_CONFIG
AUR_ADC_IMMEDIATE
AUR_DIO_SPI_WRITE
AUR_DIO_SPI_READ
AUR_ADC_GET_CONFIG
CYPRESS_GET_DESC
VENDOR_REQUEST_end
VENDOR_REQUEST_begin

AUR_DIO_WRITE

AUR_DIO_READ

AUR_DIO_CONFIG

AUR_DIO_CONFIG_QUERY

AUR_CTR_READ

AUR_CTR_MODE

AUR_CTR_LOAD

AUR_CTR_MODELOAD

AUR_CTR_SELGATE

AUR_CTR_READALL

AUR_CTR_READLATCHED

AUR_CTR_COS_BULK_GATE2

AUR_CTR_PUR_FIRST

AUR_CTR_PUR_OFRQ

AUR_CTR_COS_BULK_ABORT

AUR_CTR_PUR_MFRQ

AUR_CTR_PUR_EVCT

AUR_CTR_PUR_MPUL

AUR_WDG_STATUS

AUR_DIO_WDG16_DEPREC

AUR_READBACK_GLOBAL_STATE

AUR_SAVE_GLOBAL_STATE

AUR_GEN_CLEAR_FIFO_NEXT

AUR_GEN_CLEAR_FIFO

AUR_GEN_CLEAR_FIFO_WAIT

AUR_GEN_ABORT_AND_CLEAR

AUR_WDG

AUR_OFFLINE_READWRITE

AUR_SELF_TEST_1

AUR_EEPROM_READ

AUR_EEPROM_WRITE

AUR_DAC_CONTROL

AUR_DAC_DATAPTR

AUR_DAC_DIVISOR

AUR_DAC_IMMEDIATE

AUR_GEN_STREAM_STATUS

AUR_FLASH_READWRITE

AUR_DAC_RANGE

AUR_PROBE_CALFEATURE

AUR_LOAD_BULK_CALIBRATION_BLOCK

AUR_DIO_STREAM_OPEN_OUTPUT

AUR_START_ACQUIRING_BLOCK

AUR_DIO_STREAM_OPEN_INPUT

AUR_DIO_SETCLOCKS

AUR_ADC_SET_CONFIG

AUR_ADC_IMMEDIATE

AUR_DIO_SPI_WRITE

AUR_DIO_SPI_READ

AUR_ADC_GET_CONFIG

CYPRESS_GET_DESC

VENDOR_REQUEST_end

anonymous enum

Enumerator

BITS_PER_BYTE
AI_16_MAX_COUNTS
MAX_IMM_ADCS
CAL_TABLE_WORDS
COUNTERS_PER_BLOCK
COUNTER_NUM_MODES
DAC_RESET
CYPRESS_DESC_PARAMS
CYPRESS_MAX_DESC_SIZE
AIOUSB_MAX_NAME_SIZE
EEPROM_SERIAL_NUMBER_ADDRESS
EEPROM_CUSTOM_BASE_ADDRESS
EEPROM_CUSTOM_MIN_ADDRESS
EEPROM_CUSTOM_MAX_ADDRESS
AD_CONFIG_REGISTERS
AD_MUX_CONFIG_REGISTERS
USB_WRITE_TO_DEVICE
USB_READ_FROM_DEVICE
USB_BULK_WRITE_ENDPOINT
USB_BULK_READ_ENDPOINT

enum **ADCalMode**

Enumerator

ADCalMode_begin
AD_CAL_MODE_NORMAL
AD_CAL_MODE_GROUND
AD_CAL_MODE_REFERENCE
AD_CAL_MODE_BIP_GROUND
AD_CAL_MODE_HIGH_REF
ADCalMode_end
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
AIOCommandCode_begin
GENERIC_DOSOMETHING_PLACEHOLDER
AIO_CONTINUE_RUNNING
AIO_TERMINATE_CALLBACK
AIOCommandCode_end

34.142 lib/ndkbuild/jni/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_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,...)
- #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, 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, 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_NORMAL, AIOCONTINUOUS_BUF_OVERRIDE,
  AIOContinuousBufMode_end, AIOContinuousBufMode_begin = (0 -1), AIOCONTINUOUS_BUF_ALLORNONE, AIOCONTINUOUS_BUF_NORMAL,
  AIOCONTINUOUS_BUF_OVERRIDE, AIOContinuousBufMode_end }
- enum { MAX_USB_DEVICES = 32 }
- enum AIOUSB_BOOL_VAL { AIOUSB_FALSE = 0, AIOUSB_TRUE = 1, 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,
ProductIDS_begin = (0 -1), ACCES_VENDOR_ID = 0x1605, USB_DA12_8A_REV_A = 0xC001, USB_DA12_8A = 0xC002,
USB_DA12_8E = 0xC003, USB_DIO_32 = 0x8001, USB_DIO_32I = 0x8004, USB_DIO_48 = 0x8002,
USB_DIO_96 = 0x8003, USB_DIO24_CTR6 = 0x8006, USB_DI16A_REV_A1 = 0x8008, USB_DO16A_REV_A1 = 0x8009,
USB_DI16A_REV_A2 = 0x800a, USB_DIO_16H = 0x800c, USB_DI16A = 0x800d, USB_DO16A = 0x800e,
USB_DIO_16A = 0x800f, USB_IIRO_16 = 0x8010, USB_II_16 = 0x8011, USB_RO_16 = 0x8012,
USB_IIRO_8 = 0x8014, USB_II_8 = 0x8015, USB_IIRO_4 = 0x8016, USB_IDIO_16 = 0x8018,
USB_II_16_OLD = 0x8019, USB_IDO_16 = 0x801a, USB_IDIO_8 = 0x801c, USB_II_8_OLD = 0x801d,
USB_IDIO_4 = 0x801e, USB_CTR_15 = 0x8020, USB_IIRO4_2SM = 0x8030, USB_IIRO4_COM = 0x8031,
USB_DIO16RO8 = 0x8032, USB_DIO48DO24 = 0x803C, USB_DIO24DO12 = 0x803D, USB_DO24 = 0x803E,
PICO_DIO16RO8 = 0x8033, USBP_II8IDO4A = 0x8036, USB_AI16_16A = 0x8040, USB_AI16_16E = 0x8041,
USB_AI12_16A = 0x8042, USB_AI12_16 = 0x8043, USB_AI12_16E = 0x8044, USB_AI16_64MA = 0x8045,
USB_AI16_64ME = 0x8046, USB_AI12_64MA = 0x8047, USB_AI12_64M = 0x8048, USB_AI12_64ME = 0x8049,
USB_AI16_32A = 0x804a, USB_AI16_32E = 0x804b, USB_AI12_32A = 0x804c, USB_AI12_32 = 0x804d,
USB_AI12_32E = 0x804e, USB_AI16_64A = 0x804f, USB_AI16_64E = 0x8050, USB_AI12_64A = 0x8051,
USB_AI12_64 = 0x8052, USB_AI12_64E = 0x8053, USB_AI16_96A = 0x8054, USB_AI16_96E = 0x8055,
USB_AI12_96A = 0x8056, USB_AI12_96 = 0x8057, USB_AI12_96E = 0x8058, USB_AI16_128A = 0x8059,
USB_AI16_128E = 0x805a, USB_AI12_128A = 0x805b, USB_AI12_128 = 0x805c, USB_AI12_128E = 0x805d,
USB_AO_ARB1 = 0x8068, USB_AO16_16A = 0x8070, USB_AO16_16 = 0x8071, USB_AO16_12A = 0x8072,
USB_AO16_12 = 0x8073, USB_AO16_8A = 0x8074, USB_AO16_8 = 0x8075, USB_AO16_4A = 0x8076,
USB_AO16_4 = 0x8077, USB_AO12_16A = 0x8078, USB_AO12_16 = 0x8079, USB_AO12_12A = 0x807a,
USB_AO12_12 = 0x807b, USB_AO12_8A = 0x807c, USB_AO12_8 = 0x807d, USB_AO12_4A = 0x807e,
USB_AO12_4 = 0x807f, USB_AIO16_16A = 0x8140, USB_AIO16_16E = 0x8141, USB_AIO12_16A = 0x8142,
USB_AIO12_16 = 0x8143, USB_AIO12_16E = 0x8144, USB_AIO16_64MA = 0x8145, USB_AIO16_64ME = 0x8146,
USB_AIO12_64MA = 0x8147, USB_AIO12_64M = 0x8148, USB_AIO12_64ME = 0x8149, USB_AIO16_32A = 0x814a,
USB_AIO16_32E = 0x814b, USB_AIO12_32A = 0x814c, USB_AIO12_32 = 0x814d, USB_AIO12_32E = 0x814e,
USB_AIO16_64A = 0x814f, USB_AIO16_64E = 0x8150, USB_AIO12_64A = 0x8151, USB_AIO12_64 = 0x8152,
USB_AIO12_64E = 0x8153, USB_AIO16_96A = 0x8154, USB_AIO16_96E = 0x8155, USB_AIO12_96A = 0x8156,
USB_AIO12_96 = 0x8157, USB_AIO12_96E = 0x8158, USB_AIO16_128A = 0x8159, USB_AIO16_128E = 0x815a,
USB_AIO12_128A = 0x815b, USB_AIO12_128 = 0x815c, USB_AIO12_128E = 0x815d, ProductIDS_end }

- enum { diFirst = 0xFFFFFFFEul, diOnly = 0xFFFFFFFDul, diNone = 0xFFFFFFFFul }
- enum DACRange {
  DACRange_begin = (0 -1), DAC_RANGE_0_5V, DAC_RANGE_5V, DAC_RANGE_0_10V,
  DAC_RANGE_10V, DACRange_end, 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_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, 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, AIOU-

SB_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, 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, AD_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, 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_READLATCHED = 0x26, AUR_CTR_COS_BULK_GATE2 = 0x27, AUR_CTR_PUR_FIRST = 0x28, AUR_CTR_PUR_OFRQ = 0x28, AUR_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_STATE = 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_EEPROM_WRITE = 0XA2, AUR_DAC_CONTROL = 0xB0, AUR_DAC_DATAPTR = 0xB1, AUR_DAC_DIVISOR = 0xB2, AUR_DAC_IMMEDIATE = 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_START_ACQUIRING_BLOCK = 0xBC, AUR_DIO_STREAM_OPEN_INPUT = 0xBC,
AUR_DIO_SETCLOCKS = 0xBD, AUR_ADC_SET_CONFIG = 0xBE, AUR_ADC_IMMEDIATE = 0xBF, AUR_DIO_SPI_WRITE = 0xC0,
AUR_DIO_SPI_READ = 0xC1, AUR_ADC_GET_CONFIG = 0xD2, CYPRESS_GET_DESC = 0x06, VENDOR_REQUEST_end,
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_READLATCHED = 0x26,
AUR_CTR_COS_BULK_GATE2 = 0x27, AUR_CTR_PUR_FIRST = 0x28, AUR_CTR_PUR_OFRQ = 0x28, AUR_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_STATE = 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_EEPROM_WRITE = 0XA2,
AUR_DAC_CONTROL = 0xB0, AUR_DAC_DATAPTR = 0xB1, AUR_DAC_DIVISOR = 0xB2, AUR_DAC_IMMEDIATE = 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_START_ACQUIRING_BLOCK = 0xBC, AUR_DIO_STREAM_OPEN_INPUT = 0xBC,
AUR_DIO_SETCLOCKS = 0xBD, AUR_ADC_SET_CONFIG = 0xBE, AUR_ADC_IMMEDIATE = 0xBF, AUR_DIO_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_PARAMS = 0x0302,
  CYPRESS_MAX_DESC_SIZE = 256, AIOUSB_MAX_NAME_SIZE = 100, EEPROM_SERIAL_NUMBER_ADDRESS = 0x1DF8, EEPROM_CUSTOM_BASE_ADDRESS = 0x1E00,
  EEPROM_CUSTOM_MIN_ADDRESS = 0, EEPROM_CUSTOM_MAX_ADDRESS = 0x1FF, AD_CONFIG_REGISTERS = 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_REFERENCE = 3,
  AD_CAL_MODE_BIP_GROUND = 5, AD_CAL_MODE_HIGH_REF = 7, ADCalMode_end = 8, ADCalMode_begin = -1,
  AD_CAL_MODE_NORMAL = 0, AD_CAL_MODE_GROUND = 1, AD_CAL_MODE_REFERENCE = 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_RUNNING, AIO_TERMINATE_CALLBACK,
  AIOCommandCode_end, AIOCommandCode_begin = (0 -1), GENERIC_DOSOMETHING_PLACEHOLDER, AIO_CONTINUE_RUNNING,
  AIO_TERMINATE_CALLBACK, AIOCommandCode_end }

  Enums that govern how commands are performed and operated.

34.142.1 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_AIOCHANNEL-MASK, __VA_ARGS__)

#define AIO_ASSERT_CONFIG(*...*) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_ADCCONFIG , __VA_ARGS__)

#define AIO_ASSERT_AIOCONTBUF(*...*) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_AIOCONTINUOU-S_BUFFER, __VA_ARGS__);

#define AIO_ASSERT_AIOEITHER(*err, msg, ...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { AIOEither tmp; tmp.left = err; tmp.errmsg=strdup(msg); return tmp; }

#define AIO_ERROR(*X*) (-abs(X))

AIO_ERROR ∗ are just like the regular ASSERTIONS meaning that they argument should evaluate to true otherwise it will fail.

#define AIO_ERROR_VALID_DATA(*err, ...*) if (!(__VA_ARGS__)) { return err; }

#define AIO_ERROR_VALID_DATA_RETVAL(*err, ...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_VALID_AIORET_TYPE(*err, ...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_AIOEITHER_VALID_DATA(*err, ...*)

Value:

```
if ( !(__VA_ARGS__) ) { \
        AIOEither tmp; tmp.left = err; tmp.errmsg=NULL; return tmp; }
```

#define AIO_ERROR_VALID_DATA_W_CODE(*err, code, ...*) if (!(__VA_ARGS__)) { { code; }; return err; }

#define AIO_ERROR_VALID_DATA_WITH_CODE(*retval, err, code*) if (! (code)) { errno = -abs(err); return retval; }

#define G_STMT_START do

#define G_STMT_END while (0)

#define G_BREAKPOINT( ) G_STMT_START{ raise (SIGTRAP); }G_STMT_END

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                             \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;                 \
            goto g;                                             \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {            \
            goto g;                                             \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {    \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;                \
            goto g;                                             \
        }                                                       \
    } while (0 )
```

#define AIOUSB_ERROR_OFFSET 100

#define LIBUSB_RESULT_TO_AIOUSB_RESULT(*code*) (unsigned long)(AIOUSB_ERROR_OFFSET + -(int)(code))

#define AIOUSB_RESULT_TO_LIBUSB_RESULT(*code*) (-((int)(code) - AIOUSB_ERROR_OFFSET))

#define ROOTCLOCK 10000000

34.142.2 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**

34.142.3 Enumeration Type Documentation

enum **AIO_SCAN_TYPE**

Enumerator

AIO_PER_OVERSAMPLE
AIO_PER_CHANNEL
AIO_PER_SCANS
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
THREAD_STATUS_begin
INVALID_OBJECT
NOT_STARTED
RUNNING
WITH_DATA
TERMINATED
RUNNING_OR_WITH_DATA
JOINED
TERMINATED_OVERRUN
TERMINATING
THREAD_STATUS_end

enum **AIOContinuousBufMode**

Enumerator

AIOContinuousBufMode_begin

AIOCONTINUOUS_BUF_ALLORNONE

AIOCONTINUOUS_BUF_NORMAL

AIOCONTINUOUS_BUF_OVERRIDE

AIOContinuousBufMode_end

AIOContinuousBufMode_begin

AIOCONTINUOUS_BUF_ALLORNONE

AIOCONTINUOUS_BUF_NORMAL

AIOCONTINUOUS_BUF_OVERRIDE

AIOContinuousBufMode_end

anonymous enum

Enumerator

MAX_USB_DEVICES

enum **AIOUSB_BOOL_VAL**

other libraries often declare BOOL, TRUE and FALSE, and worse, they declare these using #define; so we sidestep that potential conflict by declaring the same types prefixed with AIOUSB_; it's ugly, but if people want to use the shorter names and they are certain they won't conflict with anything else, they can define the ENABLE_BOOL_TYPE macro

Enumerator

AIOUSB_FALSE

AIOUSB_TRUE

AIOUSB_FALSE

AIOUSB_TRUE

enum **ProductIDS**

Enumerator

ProductIDS_begin

ACCES_VENDOR_ID

USB_DA12_8A_REV_A • these product IDs are constant ∗

USB_DA12_8A

USB_DA12_8E

USB_DIO_32 • these are the product IDs after firmware is uploaded to the device; prior ∗ to uploading the firmware, the product ID is that shown below minus 0x8000

USB_DIO_32I

USB_DIO_48

USB_DIO_96

USB_DIO24_CTR6

USB_DI16A_REV_A1

USB_DO16A_REV_A1

USB_DI16A_REV_A2

USB_DIO_16H

USB_DI16A

USB_DO16A

USB_DIO_16A

USB_IIRO_16

USB_II_16

USB_RO_16

USB_IIRO_8

USB_II_8

USB_IIRO_4
USB_IDIO_16
USB_II_16_OLD
USB_IDO_16
USB_IDIO_8
USB_II_8_OLD
USB_IDIO_4
USB_CTR_15
USB_IIRO4_2SM
USB_IIRO4_COM
USB_DIO16RO8
USB_DIO48DO24
USB_DIO24DO12
USB_DO24
PICO_DIO16RO8
USBP_II8IDO4A
USB_AI16_16A
USB_AI16_16E
USB_AI12_16A
USB_AI12_16
USB_AI12_16E
USB_AI16_64MA
USB_AI16_64ME
USB_AI12_64MA
USB_AI12_64M
USB_AI12_64ME
USB_AI16_32A
USB_AI16_32E
USB_AI12_32A
USB_AI12_32
USB_AI12_32E
USB_AI16_64A
USB_AI16_64E
USB_AI12_64A
USB_AI12_64
USB_AI12_64E
USB_AI16_96A
USB_AI16_96E
USB_AI12_96A
USB_AI12_96
USB_AI12_96E
USB_AI16_128A
USB_AI16_128E
USB_AI12_128A
USB_AI12_128
USB_AI12_128E
USB_AO_ARB1
USB_AO16_16A
USB_AO16_16
USB_AO16_12A
USB_AO16_12
USB_AO16_8A
USB_AO16_8
USB_AO16_4A
USB_AO16_4

USB_AO12_16A

USB_AO12_16

USB_AO12_12A

USB_AO12_12

USB_AO12_8A

USB_AO12_8

USB_AO12_4A

USB_AO12_4

USB_AIO16_16A

USB_AIO16_16E

USB_AIO12_16A

USB_AIO12_16

USB_AIO12_16E

USB_AIO16_64MA

USB_AIO16_64ME

USB_AIO12_64MA

USB_AIO12_64M

USB_AIO12_64ME

USB_AIO16_32A

USB_AIO16_32E

USB_AIO12_32A

USB_AIO12_32

USB_AIO12_32E

USB_AIO16_64A

USB_AIO16_64E

USB_AIO12_64A

USB_AIO12_64

USB_AIO12_64E

USB_AIO16_96A

USB_AIO16_96E

USB_AIO12_96A

USB_AIO12_96

USB_AIO12_96E

USB_AIO16_128A

USB_AIO16_128E

USB_AIO12_128A

USB_AIO12_128

USB_AIO12_128E

ProductIDS_end

ProductIDS_begin

ACCES_VENDOR_ID

USB_DA12_8A_REV_A • these product IDs are constant ∗

USB_DA12_8A

USB_DA12_8E

USB_DIO_32 • these are the product IDs after firmware is uploaded to the device; prior ∗ to uploading the firmware, the product ID is that shown below minus 0x8000

USB_DIO_32I

USB_DIO_48

USB_DIO_96

USB_DIO24_CTR6

USB_DI16A_REV_A1

USB_DO16A_REV_A1

USB_DI16A_REV_A2

USB_DIO_16H

USB_DI16A

USB_DO16A

USB_DIO_16A
USB_IIRO_16
USB_II_16
USB_RO_16
USB_IIRO_8
USB_II_8
USB_IIRO_4
USB_IDIO_16
USB_II_16_OLD
USB_IDO_16
USB_IDIO_8
USB_II_8_OLD
USB_IDIO_4
USB_CTR_15
USB_IIRO4_2SM
USB_IIRO4_COM
USB_DIO16RO8
USB_DIO48DO24
USB_DIO24DO12
USB_DO24
PICO_DIO16RO8
USBP_II8IDO4A
USB_AI16_16A
USB_AI16_16E
USB_AI12_16A
USB_AI12_16
USB_AI12_16E
USB_AI16_64MA
USB_AI16_64ME
USB_AI12_64MA
USB_AI12_64M
USB_AI12_64ME
USB_AI16_32A
USB_AI16_32E
USB_AI12_32A
USB_AI12_32
USB_AI12_32E
USB_AI16_64A
USB_AI16_64E
USB_AI12_64A
USB_AI12_64
USB_AI12_64E
USB_AI16_96A
USB_AI16_96E
USB_AI12_96A
USB_AI12_96
USB_AI12_96E
USB_AI16_128A
USB_AI16_128E
USB_AI12_128A
USB_AI12_128
USB_AI12_128E
USB_AO_ARB1
USB_AO16_16A
USB_AO16_16

USB_AO16_12A

USB_AO16_12

USB_AO16_8A

USB_AO16_8

USB_AO16_4A

USB_AO16_4

USB_AO12_16A

USB_AO12_16

USB_AO12_12A

USB_AO12_12

USB_AO12_8A

USB_AO12_8

USB_AO12_4A

USB_AO12_4

USB_AIO16_16A

USB_AIO16_16E

USB_AIO12_16A

USB_AIO12_16

USB_AIO12_16E

USB_AIO16_64MA

USB_AIO16_64ME

USB_AIO12_64MA

USB_AIO12_64M

USB_AIO12_64ME

USB_AIO16_32A

USB_AIO16_32E

USB_AIO12_32A

USB_AIO12_32

USB_AIO12_32E

USB_AIO16_64A

USB_AIO16_64E

USB_AIO12_64A

USB_AIO12_64

USB_AIO12_64E

USB_AIO16_96A

USB_AIO16_96E

USB_AIO12_96A

USB_AIO12_96

USB_AIO12_96E

USB_AIO16_128A

USB_AIO16_128E

USB_AIO12_128A

USB_AIO12_128

USB_AIO12_128E

ProductIDS_end

anonymous enum

Enumerator

diFirst

diOnly

diNone

enum DACRange

range codes passed to DACSetBoardRange()

Enumerator

DACRange_begin
DAC_RANGE_0_5V
DAC_RANGE_5V
DAC_RANGE_0_10V
DAC_RANGE_10V
DACRange_end
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
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
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
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

ADGainCode_begin

AD_GAIN_CODE_0_10V

AD_GAIN_CODE_10V

AD_GAIN_CODE_0_5V

AD_GAIN_CODE_5V

AD_GAIN_CODE_0_2V

AD_GAIN_CODE_2V

AD_GAIN_CODE_0_1V

AD_GAIN_CODE_1V

ADGainCode_end

enum **VENDOR_REQUEST**

Enumerator

VENDOR_REQUEST_begin
AUR_DIO_WRITE
AUR_DIO_READ
AUR_DIO_CONFIG
AUR_DIO_CONFIG_QUERY
AUR_CTR_READ
AUR_CTR_MODE
AUR_CTR_LOAD
AUR_CTR_MODELOAD
AUR_CTR_SELGATE
AUR_CTR_READALL
AUR_CTR_READLATCHED
AUR_CTR_COS_BULK_GATE2
AUR_CTR_PUR_FIRST
AUR_CTR_PUR_OFRQ
AUR_CTR_COS_BULK_ABORT
AUR_CTR_PUR_MFRQ
AUR_CTR_PUR_EVCT
AUR_CTR_PUR_MPUL
AUR_WDG_STATUS
AUR_DIO_WDG16_DEPREC
AUR_READBACK_GLOBAL_STATE
AUR_SAVE_GLOBAL_STATE
AUR_GEN_CLEAR_FIFO_NEXT
AUR_GEN_CLEAR_FIFO
AUR_GEN_CLEAR_FIFO_WAIT
AUR_GEN_ABORT_AND_CLEAR
AUR_WDG
AUR_OFFLINE_READWRITE
AUR_SELF_TEST_1
AUR_EEPROM_READ
AUR_EEPROM_WRITE
AUR_DAC_CONTROL
AUR_DAC_DATAPTR
AUR_DAC_DIVISOR
AUR_DAC_IMMEDIATE
AUR_GEN_STREAM_STATUS
AUR_FLASH_READWRITE
AUR_DAC_RANGE
AUR_PROBE_CALFEATURE
AUR_LOAD_BULK_CALIBRATION_BLOCK
AUR_DIO_STREAM_OPEN_OUTPUT
AUR_START_ACQUIRING_BLOCK
AUR_DIO_STREAM_OPEN_INPUT
AUR_DIO_SETCLOCKS
AUR_ADC_SET_CONFIG
AUR_ADC_IMMEDIATE
AUR_DIO_SPI_WRITE
AUR_DIO_SPI_READ
AUR_ADC_GET_CONFIG
CYPRESS_GET_DESC
VENDOR_REQUEST_end
VENDOR_REQUEST_begin

AUR_DIO_WRITE

AUR_DIO_READ

AUR_DIO_CONFIG

AUR_DIO_CONFIG_QUERY

AUR_CTR_READ

AUR_CTR_MODE

AUR_CTR_LOAD

AUR_CTR_MODELOAD

AUR_CTR_SELGATE

AUR_CTR_READALL

AUR_CTR_READLATCHED

AUR_CTR_COS_BULK_GATE2

AUR_CTR_PUR_FIRST

AUR_CTR_PUR_OFRQ

AUR_CTR_COS_BULK_ABORT

AUR_CTR_PUR_MFRQ

AUR_CTR_PUR_EVCT

AUR_CTR_PUR_MPUL

AUR_WDG_STATUS

AUR_DIO_WDG16_DEPREC

AUR_READBACK_GLOBAL_STATE

AUR_SAVE_GLOBAL_STATE

AUR_GEN_CLEAR_FIFO_NEXT

AUR_GEN_CLEAR_FIFO

AUR_GEN_CLEAR_FIFO_WAIT

AUR_GEN_ABORT_AND_CLEAR

AUR_WDG

AUR_OFFLINE_READWRITE

AUR_SELF_TEST_1

AUR_EEPROM_READ

AUR_EEPROM_WRITE

AUR_DAC_CONTROL

AUR_DAC_DATAPTR

AUR_DAC_DIVISOR

AUR_DAC_IMMEDIATE

AUR_GEN_STREAM_STATUS

AUR_FLASH_READWRITE

AUR_DAC_RANGE

AUR_PROBE_CALFEATURE

AUR_LOAD_BULK_CALIBRATION_BLOCK

AUR_DIO_STREAM_OPEN_OUTPUT

AUR_START_ACQUIRING_BLOCK

AUR_DIO_STREAM_OPEN_INPUT

AUR_DIO_SETCLOCKS

AUR_ADC_SET_CONFIG

AUR_ADC_IMMEDIATE

AUR_DIO_SPI_WRITE

AUR_DIO_SPI_READ

AUR_ADC_GET_CONFIG

CYPRESS_GET_DESC

VENDOR_REQUEST_end

anonymous enum

Enumerator

BITS_PER_BYTE
AI_16_MAX_COUNTS
MAX_IMM_ADCS
CAL_TABLE_WORDS
COUNTERS_PER_BLOCK
COUNTER_NUM_MODES
DAC_RESET
CYPRESS_DESC_PARAMS
CYPRESS_MAX_DESC_SIZE
AIOUSB_MAX_NAME_SIZE
EEPROM_SERIAL_NUMBER_ADDRESS
EEPROM_CUSTOM_BASE_ADDRESS
EEPROM_CUSTOM_MIN_ADDRESS
EEPROM_CUSTOM_MAX_ADDRESS
AD_CONFIG_REGISTERS
AD_MUX_CONFIG_REGISTERS
USB_WRITE_TO_DEVICE
USB_READ_FROM_DEVICE
USB_BULK_WRITE_ENDPOINT
USB_BULK_READ_ENDPOINT

enum **ADCalMode**

Enumerator

ADCalMode_begin
AD_CAL_MODE_NORMAL
AD_CAL_MODE_GROUND
AD_CAL_MODE_REFERENCE
AD_CAL_MODE_BIP_GROUND
AD_CAL_MODE_HIGH_REF
ADCalMode_end
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
AIOCommandCode_begin
GENERIC_DOSOMETHING_PLACEHOLDER
AIO_CONTINUE_RUNNING
AIO_TERMINATE_CALLBACK
AIOCommandCode_end

34.143 lib/aiousb.h File Reference

```
#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"
```

34.144 lib/ndkbuild/jni/aiousb.h File Reference

```
#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"
```

34.145 lib/AIOUSB_ADC.c File Reference

```
#include "AIOUSB_ADC.h"
#include "AIOUSB_CTR.h"
#include "AIOTypes.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
```

Macros

- #define DEVICE_SAMPLE_BUFFER_SIZE 1024
- #define STREAMING_PNA_DEFINITIONS

Functions

- AIORET_TYPE adc_get_bulk_data (ADCConfigBlock *config, USBDevice *usb, unsigned char endpoint, unsigned char *data, int datasize, int *bytes, unsigned timeout)

- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- unsigned long * ADC_GetConfigSize (ADConfigBlock *config)
- unsigned char * ADC_GetConfigRegisters (ADConfigBlock *config)
- AIORET_TYPE ADC_ReadADConfigBlock (unsigned long DeviceIndex, ADConfigBlock *config)
- unsigned long ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- unsigned long WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ADC_Acquire_Reference_Counts (unsigned long DeviceIndex, double *groundCounts, double *referenceCounts)

  Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.
- PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[])

  Performs a scan and averages the voltage values.
- PRIVATE unsigned long AIOUSB_ArrayCountsToVolts (unsigned long DeviceIndex, int startChannel, int numChannels, const unsigned short counts[], double volts[])
- PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long DeviceIndex, int startChannel, int numChannels, const double volts[], unsigned short counts[])
- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double *singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double *pBuf)

  Preferred way to get immediate scan readings.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short *pBuf)

  This simple function takes one scan of A/D data, in counts.
- unsigned long ADC_GetConfig (unsigned long DeviceIndex, unsigned char *ConfigBuf, unsigned long *ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- int adcblock_valid_trigger_settings (ADConfigBlock *config)
- int adcblock_valid_channel_settings (ADConfigBlock *config, int ADCMUXChannels)
- unsigned long valid_config_block (ADConfigBlock *config)
- int adcblock_valid_size (ADConfigBlock *config)
- unsigned long ADC_SetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)
- unsigned long ADC_CopyConfig (unsigned long DeviceIndex, ADConfigBlock *config)

  Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.
- unsigned long ADC_RangeAll (unsigned long DeviceIndex, unsigned char *pGainCodes, unsigned long bSingleEnded)
- unsigned long ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- unsigned long ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- unsigned long ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- unsigned long ADC_SetCal (unsigned long DeviceIndex, const char *CalFileName)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long productID)
- unsigned long ADC_QueryCal (unsigned long DeviceIndex)
- unsigned long ADC_Initialize (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize, const char *CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- unsigned long ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void *pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIOBuf * CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long ADC_BulkPoll (unsigned long DeviceIndex, unsigned long *BytesLeft)
- unsigned long ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- unsigned char * ADC_GetADConfigBlock_Registers (ADConfigBlock *config)
- AIORESULT ADC_ClearFastITConfig (unsigned long DeviceIndex)

  Frees memory associated with the FastConfig Config blocks.

- unsigned long ADC_CreateADBuf (AIOUSBDevice *const deviceDesc, int size)
- void ADC_ClearADBuf (AIOUSBDevice *deviceDesc)
- unsigned long ADC_InitFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_ResetFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- void ADC_Debug_Register_Settings (ADConfigBlock *config)

  Just a debugging function for listing all attributes of a config object.
- char * ADConfigBlockToYAML (ADConfigBlock *config)
- unsigned long ADC_GetFastITScanV (unsigned long DeviceIndex, double *pData)
- unsigned long ADC_GetITScanV (unsigned long DeviceIndex, double *pBuf)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- void AIOUSB_Copy_Config_Block (ADConfigBlock *to, ADConfigBlock *from)
- unsigned long AIOUSB_Validate_ADC_Device (unsigned long DeviceIndex)
- double GetHiRef (unsigned long deviceIndex)
- void DoLoadCalTable (unsigned short *const calTable, unsigned long DeviceIndex, double groundCounts, double referenceCounts)

  Loads the Cal table for Automatic internal calibration.
- AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock *config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ConfigureAndBulkAcquire (unsigned long DeviceIndex, ADConfigBlock *config)
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[], const char *saveFileName)

  Performs automatic calibration of the ADC.
- void AIOUSB_SetRegister (ADConfigBlock *cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock *cb, unsigned int Register)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock *config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock *config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock *config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock *config, unsigned channel)
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock *config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock *config, unsigned calMode)
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock *config, unsigned triggerMode)
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock *config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock *config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock *config, unsigned overSample)
- unsigned long AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[], int numPoints, unsigned short returnCalTable[], const char *saveFileName)

Variables

- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- int dRef = 3

34.145.1 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;
```

34.145.2 Function Documentation

AIORET_TYPE adc_get_bulk_data (**ADCConfigBlock** ∗ *config,* **USBDevice** ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

unsigned long ADC_ResetDevice (unsigned long *DeviceIndex*)

unsigned long∗ ADC_GetConfigSize (**ADConfigBlock** ∗ *config*)

unsigned char∗ ADC_GetConfigRegisters (**ADConfigBlock** ∗ *config*)

AIORET_TYPE ADC_ReadADConfigBlock (unsigned long *DeviceIndex,* **ADConfigBlock** ∗ *config*)

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

Parameters

DeviceIndex	
forceRead	

unsigned long WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ADC_Acquire_Reference_Counts (unsigned long *DeviceIndex,* double ∗ *groundCounts,* double ∗ *referenceCounts*)

Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.

Parameters

DeviceIndex	
groundCounts	
referenceCounts	

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

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment

The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

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

Parameters

DeviceIndex	
startChannel	
numChannels	
counts	
volts	

Returns

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

Parameters

DeviceIndex	
startChannel	
numChannels	
volts	
counts	

Returns

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

Parameters

DeviceIndex	
channel	
volts	

Returns

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

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short * *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

unsigned long ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char * *ConfigBuf,* unsigned long * *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

int adcblock_valid_trigger_settings (ADConfigBlock * *config*)

int adcblock_valid_channel_settings (ADConfigBlock * *config,* int *ADCMUXChannels*)

unsigned long valid_config_block (ADConfigBlock * *config*)

int adcblock_valid_size (ADConfigBlock * *config*)

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

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

unsigned long ADC_CopyConfig (unsigned long *DeviceIndex,* ADConfigBlock * *config*)

Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.

Parameters

DeviceIndex	
config	

Returns

validate settings

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

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

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

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

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

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

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

Parameters

DeviceIndex	
Oversample	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

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

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

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

Parameters

DeviceIndex	
CalFileName	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *productID*)

unsigned long ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

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

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

unsigned long ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

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

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIOBuf* CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

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

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

unsigned char* ADC_GetADConfigBlock_Registers (ADConfigBlock * *config*)

AIORESULT ADC_ClearFastITConfig (unsigned long *DeviceIndex*)

Frees memory associated with the FastConfig Config blocks.

Use this call after you are done using the ADC_FastIT* Functions

Parameters

DeviceIndex	

unsigned long ADC_CreateADBuf (AIOUSBDevice *const *deviceDesc,* int *size*)

void ADC_ClearADBuf (AIOUSBDevice * *deviceDesc*)

unsigned long ADC_InitFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

void ADC_Debug_Register_Settings (ADConfigBlock * *config*)

Just a debugging function for listing all attributes of a config object.

char* ADConfigBlockToYAML (ADConfigBlock * *config*)

```
* ---
* config:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
```

```
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   calibration: Normal
*   trigger:
*     edge: falling edge
*     scan: all channels
*     type: external
*
```

unsigned long ADC_GetFastITScanV (unsigned long *DeviceIndex,* double * *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

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

Parameters

DeviceIndex	
pBuf	

Returns

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

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

Parameters

DeviceIndex	
discard	

Returns

void AIOUSB_Copy_Config_Block (ADConfigBlock * *to,* ADConfigBlock * *from*)

unsigned long AIOUSB_Validate_ADC_Device (unsigned long *DeviceIndex*)

double GetHiRef (unsigned long *deviceIndex*)

Parameters

deviceIndex	

Returns

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

Loads the Cal table for Automatic internal calibration.

Parameters

calTable	
DeviceIndex	
groundCounts	
referenceCounts	

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

Parameters

config	
channel	
gainCode	

AIORET_TYPE ConfigureAndBulkAcquire (unsigned long *DeviceIndex,* ADConfigBlock ∗ *config*)

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex,* AIOUSB_BOOL *autoCal,* unsigned short *returnCalTable[],* const char ∗ *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

void AIOUSB_SetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register*)

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock ∗ *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock ∗ *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock ∗ *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock ∗ *config,* unsigned *channel*)

Parameters

config	
channel	

Returns

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock * *config*, unsigned *channel*, AIOUSB_BOOL *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config*, unsigned *calMode*)

Parameters

config	
calMode	

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock * *config*, unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const ADConfigBlock * *config*)

unsigned AIOUSB_GetEndChannel (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock * *config*, unsigned *startChannel*, unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock * *config*, unsigned *overSample*)

unsigned long AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex*, const double *points[]*, int *numPoints*, unsigned short *returnCalTable[]*, const char * *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
```

```
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                           ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

34.145.3 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

34.146 lib/ndkbuild/jni/AIOUSB_ADC.c File Reference

```
#include "AIOUSB_ADC.h"
#include "AIOUSB_CTR.h"
#include "AIOTypes.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
```

Macros

- #define DEVICE_SAMPLE_BUFFER_SIZE 1024
- #define STREAMING_PNA_DEFINITIONS

Functions

- AIORET_TYPE adc_get_bulk_data (ADCConfigBlock ∗config, USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int datasize, int ∗bytes, unsigned timeout)
- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- unsigned long ∗ ADC_GetConfigSize (ADConfigBlock ∗config)
- unsigned char ∗ ADC_GetConfigRegisters (ADConfigBlock ∗config)
- AIORET_TYPE ADC_ReadADConfigBlock (unsigned long DeviceIndex, ADConfigBlock ∗config)
- unsigned long ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- unsigned long WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ADC_Acquire_Reference_Counts (unsigned long DeviceIndex, double ∗groundCounts, double ∗referenceCounts)

  Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.
- PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[])

  Performs a scan and averages the voltage values.
- PRIVATE unsigned long AIOUSB_ArrayCountsToVolts (unsigned long DeviceIndex, int startChannel, int numChannels, const unsigned short counts[], double volts[])
- PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long DeviceIndex, int startChannel, int numChannels, const double volts[], unsigned short counts[])
- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double ∗singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double ∗pBuf)

  Preferred way to get immediate scan readings.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short ∗pBuf)

  This simple function takes one scan of A/D data, in counts.
- unsigned long ADC_GetConfig (unsigned long DeviceIndex, unsigned char ∗ConfigBuf, unsigned long ∗ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- int adcblock_valid_trigger_settings (ADConfigBlock ∗config)
- int adcblock_valid_channel_settings (ADConfigBlock ∗config, int ADCMUXChannels)
- unsigned long valid_config_block (ADConfigBlock ∗config)
- int adcblock_valid_size (ADConfigBlock ∗config)
- unsigned long ADC_SetConfig (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize)
- unsigned long ADC_CopyConfig (unsigned long DeviceIndex, ADConfigBlock ∗config)

  Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.
- unsigned long ADC_RangeAll (unsigned long DeviceIndex, unsigned char ∗pGainCodes, unsigned long bSingleEnded)
- unsigned long ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- unsigned long ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)

- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- unsigned long ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- unsigned long ADC_SetCal (unsigned long DeviceIndex, const char *CalFileName)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long productID)
- unsigned long ADC_QueryCal (unsigned long DeviceIndex)
- unsigned long ADC_Initialize (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize, const char *CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- unsigned long ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void *pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIOBuf * CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long ADC_BulkPoll (unsigned long DeviceIndex, unsigned long *BytesLeft)
- unsigned long ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- unsigned char * ADC_GetADConfigBlock_Registers (ADConfigBlock *config)
- AIORESULT ADC_ClearFastITConfig (unsigned long DeviceIndex)

  Frees memory associated with the FastConfig Config blocks.
- unsigned long ADC_CreateADBuf (AIOUSBDevice *const deviceDesc, int size)
- void ADC_ClearADBuf (AIOUSBDevice *deviceDesc)
- unsigned long ADC_InitFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_ResetFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- void ADC_Debug_Register_Settings (ADConfigBlock *config)

  Just a debugging function for listing all attributes of a config object.
- char * ADConfigBlockToYAML (ADConfigBlock *config)
- unsigned long ADC_GetFastITScanV (unsigned long DeviceIndex, double *pData)
- unsigned long ADC_GetITScanV (unsigned long DeviceIndex, double *pBuf)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- void AIOUSB_Copy_Config_Block (ADConfigBlock *to, ADConfigBlock *from)
- unsigned long AIOUSB_Validate_ADC_Device (unsigned long DeviceIndex)
- double GetHiRef (unsigned long deviceIndex)
- void DoLoadCalTable (unsigned short *const calTable, unsigned long DeviceIndex, double groundCounts, double referenceCounts)

  Loads the Cal table for Automatic internal calibration.
- AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock *config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ConfigureAndBulkAcquire (unsigned long DeviceIndex, ADConfigBlock *config)
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[], const char *saveFileName)

  Performs automatic calibration of the ADC.
- void AIOUSB_SetRegister (ADConfigBlock *cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock *cb, unsigned int Register)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock *config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock *config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock *config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock *config, unsigned channel)
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock *config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock *config, unsigned calMode)
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock *config, unsigned triggerMode)
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock *config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock *config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock *config, unsigned overSample)
- unsigned long AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[], int numPoints, unsigned short returnCalTable[], const char *saveFileName)

Variables

- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- int dRef = 3

34.146.1 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;
```

34.146.2 Function Documentation

AIORET_TYPE adc_get_bulk_data (**ADCConfigBlock** ∗ *config,* **USBDevice** ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

unsigned long ADC_ResetDevice (unsigned long *DeviceIndex*)

unsigned long∗ ADC_GetConfigSize (**ADConfigBlock** ∗ *config*)

unsigned char∗ ADC_GetConfigRegisters (**ADConfigBlock** ∗ *config*)

AIORET_TYPE ADC_ReadADConfigBlock (unsigned long *DeviceIndex,* **ADConfigBlock** ∗ *config*)

unsigned long ReadConfigBlock (unsigned long *DeviceIndex,* **AIOUSB_BOOL** *forceRead*)

Parameters

DeviceIndex	
forceRead	

unsigned long WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ADC_Acquire_Reference_Counts (unsigned long *DeviceIndex,* double ∗ *groundCounts,* double ∗ *referenceCounts*)

Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.

Parameters

DeviceIndex	
groundCounts	
referenceCounts	

PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment
The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

PRIVATE unsigned long AIOUSB_ArrayCountsToVolts (unsigned long *DeviceIndex,* int *startChannel,* int *numChannels,* const unsigned short *counts[],* double *volts[]*)

Parameters

DeviceIndex	
startChannel	
numChannels	
counts	
volts	

Returns

PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long *DeviceIndex,* int *startChannel,* int *numChannels,* const double *volts[],* unsigned short *counts[]*)

Parameters

DeviceIndex	
startChannel	
numChannels	
volts	
counts	

Returns

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double * *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short * *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

unsigned long ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char * *ConfigBuf,* unsigned long * *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

int adcblock_valid_trigger_settings (ADConfigBlock * *config*)

int adcblock_valid_channel_settings (ADConfigBlock * *config,* int *ADCMUXChannels*)

unsigned long valid_config_block (ADConfigBlock * *config*)

int adcblock_valid_size (ADConfigBlock * *config*)

unsigned long ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

unsigned long ADC_CopyConfig (unsigned long *DeviceIndex,* ADConfigBlock * *config*)

Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.

Parameters

DeviceIndex	
config	

Returns

validate settings

unsigned long ADC_RangeAll (unsigned long *DeviceIndex,* unsigned char * *pGainCodes,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	

bSingleEnded	

Returns

unsigned long ADC_Range1 (unsigned long *DeviceIndex,* unsigned long *ADChannel,* unsigned char *GainCode,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

unsigned long ADC_ADMode (unsigned long *DeviceIndex,* unsigned char *TriggerMode,* unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex,* unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

unsigned long ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

unsigned long ADC_SetCal (unsigned long *DeviceIndex,* const char * *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *productID*)

unsigned long ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long ADC_Initialize (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize,* const char * *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

unsigned long ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIOBuf∗ CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf ∗ new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long ∗ *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

unsigned long ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

unsigned char∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗ *config*)

AIORESULT ADC_ClearFastITConfig (unsigned long *DeviceIndex*)

Frees memory associated with the FastConfig Config blocks.

Use this call after you are done using the ADC_FastIT∗ Functions

Parameters

DeviceIndex	

unsigned long ADC_CreateADBuf (AIOUSBDevice ∗const *deviceDesc,* int *size*)

void ADC_ClearADBuf (AIOUSBDevice ∗ *deviceDesc*)

unsigned long ADC_InitFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

void ADC_Debug_Register_Settings (ADConfigBlock * *config*)

Just a debugging function for listing all attributes of a config object.

char* ADConfigBlockToYAML (ADConfigBlock * *config*)

```
* ---
* config:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   calibration: Normal
*   trigger:
*     edge: falling edge
*     scan: all channels
*     type: external
*
```

unsigned long ADC_GetFastITScanV (unsigned long *DeviceIndex,* double * *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

unsigned long ADC_GetITScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex,* AIOUSB_BOOL *discard*)

Parameters

DeviceIndex	
discard	

Returns

void AIOUSB_Copy_Config_Block (ADConfigBlock * *to,* ADConfigBlock * *from*)

unsigned long AIOUSB_Validate_ADC_Device (unsigned long *DeviceIndex*)

double GetHiRef (unsigned long *deviceIndex*)

Parameters

deviceIndex	

Returns

void DoLoadCalTable (unsigned short *const *calTable,* unsigned long *DeviceIndex,* double *groundCounts,* double *referenceCounts*)

Loads the Cal table for Automatic internal calibration.

Parameters

calTable	
DeviceIndex	
groundCounts	
referenceCounts	

AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock * *config,* unsigned long *channel,* unsigned char *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE ConfigureAndBulkAcquire (unsigned long *DeviceIndex,* ADConfigBlock * *config*)

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex,* AIOUSB_BOOL *autoCal,* unsigned short *returnCalTable[],* const char * *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

void AIOUSB_SetRegister (ADConfigBlock * *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock * *cb,* unsigned int *Register*)

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock * *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (**const ADConfigBlock** * *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (**ADConfigBlock** * *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock * *config*, unsigned *channel*)

Parameters

config	
channel	

Returns

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock * *config*, unsigned *channel*, AIOUSB_BOOL *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config*, unsigned *calMode*)

Parameters

config	
calMode	

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock * *config*, unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const ADConfigBlock * *config*)

unsigned AIOUSB_GetEndChannel (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock * *config*, unsigned *startChannel*, unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock * *config*, unsigned *overSample*)

unsigned long AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex*, const double *points[]*, int *numPoints*, unsigned short *returnCalTable[]*, const char * *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

34.146.3 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

34.147 lib/AIOUSB_ADC.h File Reference

```
#include "AIOTypes.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "USBDevice.h"
```

Functions

- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double *voltages)

  Preferred way to get immediate scan readings.
- AIORESULT ADC_RangeAll (unsigned long DeviceIndex, unsigned char *pGainCodes, unsigned long bSingleEnded)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double *singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short *pBuf)

  This simple function takes one scan of A/D data, in counts.
- AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[])

  Performs a scan and averages the voltage values.
- AIORESULT ADC_GetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_SetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)
- AIORESULT ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- AIORESULT ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- AIORESULT ADC_SetCal (unsigned long DeviceIndex, const char *CalFileName)
- AIORESULT ADC_QueryCal (unsigned long DeviceIndex)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long ProductID)
- AIORESULT ADC_Initialize (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize, const char *CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void *pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkPoll (unsigned long DeviceIndex, unsigned long *BytesLeft)
- AIORESULT ADC_InitFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- AIORESULT ADC_ResetFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- AIORESULT ADC_GetFastITScanV (unsigned long DeviceIndex, double *pData)
- AIORESULT ADC_GetITScanV (unsigned long DeviceIndex, double *pBuf)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- AIORESULT WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock *config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock *config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock *config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock *config, unsigned channel)
- AIORESULT AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[], int numPoints, unsigned short returnCalTable[], const char *saveFileName)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetAllGainCodeAndDiffMode instead")
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock *config, unsigned channel, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetDifferentialMode")

- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock *config) ACCES_DEPRECATED("Please use ADCConfigBlockGetCalMode")
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock *config, unsigned calMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetCalMode")
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock *config, unsigned overSample) ACCES_DEPRECATED("Please use ADCConfigBlockSetOversample")
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock *config) ACCES_DEPRECATED("Please use ADCConfigBlockGetOversample")
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock *config) ACCES_DEPRECATED("Please use ADCConfigBlockGetTriggerMode")
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock *config, unsigned triggerMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetTriggerMode")
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock *config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock *config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock *config, unsigned startChannel, unsigned endChannel)
- unsigned long AIOUSB_SetStreamingBlockSize (unsigned long DeviceIndex, unsigned long BlockSize)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- AIOBuf * CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[], const char *saveFileName)

  Performs automatic calibration of the ADC.
- unsigned char * ADC_GetADConfigBlock_Registers (ADConfigBlock *config)
- void AIOUSB_SetRegister (ADConfigBlock *cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock *cb, unsigned int Register)

34.147.1 Function Documentation

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex*, double * *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORESULT ADC_RangeAll (unsigned long *DeviceIndex*, unsigned char * *pGainCodes*, unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double * *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short * *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment
The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

AIORESULT ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char * *ConfigBuf,* unsigned long * *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

AIORESULT ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

validate settings

AIORESULT ADC_Range1 (unsigned long *DeviceIndex,* unsigned long *ADChannel,* unsigned char *GainCode,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

AIORESULT ADC_ADMode (unsigned long *DeviceIndex,* unsigned char *TriggerMode,* unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

AIORESULT ADC_SetCal (unsigned long *DeviceIndex,* const char * *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIORESULT ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *ProductID*)

AIORESULT ADC_Initialize (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize,* const char * *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

AIORESULT ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIORESULT ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

AIORESULT ADC_InitFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

AIORESULT ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

AIORESULT ADC_GetFastITScanV (unsigned long *DeviceIndex,* double * *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

AIORESULT ADC_GetITScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex,* unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

AIORESULT WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ReadConfigBlock (unsigned long *DeviceIndex,* AIOUSB_BOOL *forceRead*)

Parameters

DeviceIndex	
forceRead	

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock * *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock * *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock * *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock * *config,* unsigned *channel*)

Parameters

config	
channel	

Returns

AIORESULT AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex,* const double *points[],* int *numPoints,* unsigned short *returnCalTable[],* const char * *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
```

```
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_SetDifferentialMode (**ADConfigBlock** * *config,* unsigned *channel,* **AIOUSB_BOOL** *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (**const ADConfigBlock** ∗ *config*)

AIORET_TYPE AIOUSB_SetCalMode (**ADConfigBlock** ∗ *config,* unsigned *calMode*)

Parameters

config	
calMode	

AIORET_TYPE AIOUSB_SetOversample (**ADConfigBlock** ∗ *config,* unsigned *overSample*)

AIORET_TYPE AIOUSB_GetOversample (**ADConfigBlock** ∗ *config*)

unsigned AIOUSB_GetTriggerMode (**const ADConfigBlock** ∗ *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (**ADConfigBlock** ∗ *config,* unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (**const ADConfigBlock** ∗ *config*)

unsigned AIOUSB_GetEndChannel (**const ADConfigBlock** ∗ *config*)

AIORET_TYPE AIOUSB_SetScanRange (**ADConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex,* unsigned long *BlockSize*)

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex,* **AIOUSB_BOOL** *discard*)

Parameters

DeviceIndex	
discard	

Returns

AIOBuf* CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex,* AIOUSB_BOOL *autoCal,* unsigned short *returnCalTable[],* const char * *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

unsigned char* ADC_GetADConfigBlock_Registers (ADConfigBlock * *config*)

void AIOUSB_SetRegister (ADConfigBlock * *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock * *cb,* unsigned int *Register*)

34.148 lib/ndkbuild/jni/AIOUSB_ADC.h File Reference

```
#include "AIOTypes.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "USBDevice.h"
```

Functions

- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double *voltages)

  Preferred way to get immediate scan readings.
- AIORESULT ADC_RangeAll (unsigned long DeviceIndex, unsigned char *pGainCodes, unsigned long bSingleEnded)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double *singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short *pBuf)

  This simple function takes one scan of A/D data, in counts.
- AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[])

  Performs a scan and averages the voltage values.
- AIORESULT ADC_GetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.

- AIORESULT ADC_SetConfig (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize)
- AIORESULT ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- AIORESULT ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- AIORESULT ADC_SetCal (unsigned long DeviceIndex, const char ∗CalFileName)
- AIORESULT ADC_QueryCal (unsigned long DeviceIndex)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long ProductID)
- AIORESULT ADC_Initialize (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize, const char ∗CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void ∗pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkPoll (unsigned long DeviceIndex, unsigned long ∗BytesLeft)
- AIORESULT ADC_InitFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- AIORESULT ADC_ResetFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- AIORESULT ADC_GetFastITScanV (unsigned long DeviceIndex, double ∗pData)
- AIORESULT ADC_GetITScanV (unsigned long DeviceIndex, double ∗pBuf)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- AIORESULT WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock ∗config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock ∗config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock ∗config, unsigned channel)
- AIORESULT AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[], int numPoints, unsigned short returnCalTable[], const char ∗saveFileName)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetAllGainCodeAndDiffMode instead")
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetDifferentialMode")
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetCalMode")
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock ∗config, unsigned calMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetCalMode")
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗config, unsigned overSample) ACCES_DEPRECATED("Please use ADCConfigBlockSetOversample")
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetOversample")
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetTriggerMode")
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock ∗config, unsigned triggerMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetTriggerMode")
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock ∗config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- unsigned long AIOUSB_SetStreamingBlockSize (unsigned long DeviceIndex, unsigned long BlockSize)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- AIOBuf ∗ CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[], const char ∗saveFileName)

  Performs automatic calibration of the ADC.
- unsigned char ∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗config)
- void AIOUSB_SetRegister (ADConfigBlock ∗cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock ∗cb, unsigned int Register)

34.148.1 Function Documentation

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORESULT ADC_RangeAll (unsigned long *DeviceIndex,* unsigned char * *pGainCodes,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double * *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to

scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short * *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment
The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB,

in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

AIORESULT ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char * *ConfigBuf,* unsigned long * *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

AIORESULT ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

validate settings

AIORESULT ADC_Range1 (unsigned long *DeviceIndex,* unsigned long *ADChannel,* unsigned char *GainCode,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

AIORESULT ADC_ADMode (unsigned long *DeviceIndex,* unsigned char *TriggerMode,* unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

AIORESULT ADC_SetCal (unsigned long *DeviceIndex,* const char * *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIORESULT ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *ProductID*)

AIORESULT ADC_Initialize (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize,* const char * *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

AIORESULT ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIORESULT ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

AIORESULT ADC_InitFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

AIORESULT ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

AIORESULT ADC_GetFastITScanV (unsigned long *DeviceIndex,* double * *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

AIORESULT ADC_GetITScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex,* unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

AIORESULT WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ReadConfigBlock (unsigned long *DeviceIndex,* AIOUSB_BOOL *forceRead*)

Parameters

DeviceIndex	
forceRead	

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock * *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock * *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock * *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock * *config*, unsigned *channel*)

Parameters

config	
channel	

Returns

AIORESULT AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex*, const double *points[]*, int *numPoints*, unsigned short *returnCalTable[]*, const char * *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range;

similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock * *config,* unsigned *channel,* AIOUSB_BOOL *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config,* unsigned *calMode*)

Parameters

config	
calMode	

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock * *config*, unsigned *overSample*)

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock * *config*)

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock * *config*, unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const ADConfigBlock * *config*)

unsigned AIOUSB_GetEndChannel (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock * *config*, unsigned *startChannel*, unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex*, unsigned long *BlockSize*)

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex*, AIOUSB_BOOL *discard*)

Parameters

DeviceIndex	
discard	

Returns

AIOBuf* CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex*, AIOUSB_BOOL *autoCal*, unsigned short *returnCalTable[]*, const char * *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

unsigned char* ADC_GetADConfigBlock_Registers (ADConfigBlock * *config*)

void AIOUSB_SetRegister (ADConfigBlock * *cb*, unsigned int *Register*, unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock * *cb*, unsigned int *Register*)

34.149 lib/AIOUSB_Core.c File Reference

```
#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 []

34.149.1 Macro Definition Documentation

#define BACKTRACE_DEBUG(*n*) {}

#define AIOUSB_ENABLE_MUTEX

34.149.2 Function Documentation

AIOUSB_BOOL AIOUSB_Lock (void)

Notes on mutual exclusion / threading:

- Our mutual exclusion scheme is *not* intended to be bulletproof. It's primarily intended to ensure mutually exclusive access to deviceTable[] and other global variables. It does NOT ensure mutually exclusive access to the USB bus. In fact, we want to permit threads to communicate with multiple devices simultaneously, to the extent possible with USB.

- Nor does this scheme prevent multiple threads from altering the configuration of the same device or communicating with the same device. In other words, it's entirely possible for one thread to configure and communicate with a device, only to have another thread come along and to the same. It's up to the users of this library to ensure that such a scenario doesn't occur.

- This library does seek to permit one thread to control one device, and another thread to control another device. Each thread may then safely communicate with its own device and alter the portion of deviceTable[] that pertains to its device.

- Our mutual exclusion scheme also permits two threads to cooperate in the operation of a single device, such as in cases where a background thread does the actual work and the foreground thread monitors the progress. In such a case, the background thread might update a status variable which the foreground thread monitors. This form of resource sharing is supported by our mutual exclusion scheme.

AIOUSB_BOOL AIOUSB_UnLock (void)

AIORET_TYPE AIOUSB_ResetChip (unsigned long *DeviceIndex*)

unsigned long AIOUSB_Validate_Lock (unsigned long * *DeviceIndex*)

unsigned long AIOUSB_Validate (unsigned long * *DeviceIndex*)

unsigned long ResolveDeviceIndex (unsigned long *DeviceIndex*)

DeviceDescriptor* DeviceTableAtIndex (unsigned long *DeviceIndex*)

DeviceDescriptor* DeviceTableAtIndex_Lock (unsigned long *DeviceIndex*)

Todo Replace AIOUSB_Lock() with thread safe lock on a per device index basis
Insert correct error messages into global error string in case of failure

DeviceDescriptor* AIOUSB_GetDevice (unsigned long *DeviceIndex*)

ADConfigBlock* AIOUSB_GetConfigBlock (DeviceDescriptor * *dev*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex*, unsigned long *BlockSize*)

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex*, FIFO_Method *Method*)

const char* AIOUSB_GetVersion (void)

const char* AIOUSB_GetVersionDate (void)

AIORESULT AIOUSB_GetMiscClock (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_SetMiscClock (unsigned long *DeviceIndex*, double *clockHz*)

unsigned AIOUSB_GetCommTimeout (unsigned long *DeviceIndex*)

unsigned long AIOUSB_SetCommTimeout (unsigned long *DeviceIndex*, unsigned *timeout*)

unsigned long AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock * *config*, unsigned long *DeviceIndex*, AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex*, const char * *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long GenericVendorWrite (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_written*)

Performs a generic vendor USB write.

unsigned long GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_read*)

Performs basic low level USB vendor request.

Returns

34.149.3 Variable Documentation

int aio_errno

ProductIDName productIDNameTable[]

34.150 lib/ndkbuild/jni/AIOUSB_Core.c File Reference

```
#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 []

34.150.1 Macro Definition Documentation

#define BACKTRACE_DEBUG(*n*) {}

#define AIOUSB_ENABLE_MUTEX

34.150.2 Function Documentation

AIOUSB_BOOL AIOUSB_Lock (void)

Notes on mutual exclusion / threading:

- Our mutual exclusion scheme is *not* intended to be bulletproof. It's primarily intended to ensure mutually exclusive access to deviceTable[] and other global variables. It does NOT ensure mutually exclusive access to the USB bus. In fact, we want to permit threads to communicate with multiple devices simultaneously, to the extent possible with USB.

- Nor does this scheme prevent multiple threads from altering the configuration of the same device or communicating with the same device. In other words, it's entirely possible for one thread to configure and communicate with a device, only to have another thread come along and to the same. It's up to the users of this library to ensure that such a scenario doesn't occur.

- This library does seek to permit one thread to control one device, and another thread to control another device. Each thread may then safely communicate with its own device and alter the portion of deviceTable[] that pertains to its device.

- Our mutual exclusion scheme also permits two threads to cooperate in the operation of a single device, such as in cases where a background thread does the actual work and the foreground thread monitors the progress. In such a case, the background thread might update a status variable which the foreground thread monitors. This form of resource sharing is supported by our mutual exclusion scheme.

AIOUSB_BOOL AIOUSB_UnLock (**void**)

AIORET_TYPE AIOUSB_ResetChip (unsigned long *DeviceIndex*)

unsigned long AIOUSB_Validate_Lock (unsigned long * *DeviceIndex*)

unsigned long AIOUSB_Validate (unsigned long * *DeviceIndex*)

unsigned long ResolveDeviceIndex (unsigned long *DeviceIndex*)

DeviceDescriptor* DeviceTableAtIndex (unsigned long *DeviceIndex*)

DeviceDescriptor* DeviceTableAtIndex_Lock (unsigned long *DeviceIndex*)

Todo Replace AIOUSB_Lock() with thread safe lock on a per device index basis
Insert correct error messages into global error string in case of failure

DeviceDescriptor* AIOUSB_GetDevice (unsigned long *DeviceIndex*)

ADConfigBlock* AIOUSB_GetConfigBlock (**DeviceDescriptor** * *dev*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex,* unsigned long *BlockSize*)

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex,* **FIFO_Method** *Method*)

const char* AIOUSB_GetVersion (**void**)

const char* AIOUSB_GetVersionDate (**void**)

AIORESULT AIOUSB_GetMiscClock (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_SetMiscClock (unsigned long *DeviceIndex,* double *clockHz*)

unsigned AIOUSB_GetCommTimeout (unsigned long *DeviceIndex*)

unsigned long AIOUSB_SetCommTimeout (unsigned long *DeviceIndex,* unsigned *timeout*)

unsigned long AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_InitConfigBlock (**ADConfigBlock** * *config,* unsigned long *DeviceIndex,* **AIOUSB_BOOL** *defaults*)

Parameters

config	
DeviceIndex	
defaults	

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex,* **const** char * *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long GenericVendorWrite (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_written*)

Performs a generic vendor USB write.

unsigned long GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_read*)

Performs basic low level USB vendor request.

Returns

34.150.3 Variable Documentation

int aio_errno

ProductIDName productIDNameTable[]

34.151 lib/AIOUSB_Core.h File Reference

```
#include "AIOUSBDevice.h"
#include "libusb.h"
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
```

Data Structures

- struct BulkAcquireWorkerParams
- struct aiousboption
- struct aioerror
- struct ProductIDName
- struct ADRange

Macros

- #define PUBLIC_EXTERN extern
- #define PRIVATE
- #define PROD_NAME_SIZE 40

Typedefs

- typedef struct aiousboption AIOOption
- typedef struct aioerror AIOError

Functions

- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_ResetChip (unsigned long DeviceIndex)
- AIOUSB_BOOL AIOUSB_Lock (void)

  Notes on mutual exclusion / threading:
- AIOUSB_BOOL AIOUSB_UnLock (void)
- AIORESULT AIOUSB_InitTest (void)
- AIORESULT AIOUSB_Validate (unsigned long *DeviceIndex)
- AIORESULT AIOUSB_Validate_Lock (unsigned long *DeviceIndex)
- DeviceDescriptor * DeviceTableAtIndex (unsigned long DeviceIndex)
- DeviceDescriptor * DeviceTableAtIndex_Lock (unsigned long DeviceIndex)
- DeviceDescriptor * AIOUSB_GetDevice (unsigned long DeviceIndex)
- ADConfigBlock * AIOUSB_GetConfigBlock (DeviceDescriptor *dev)
- AIORESULT AIOUSB_SetMiscClock (unsigned long DeviceIndex, double clockHz)
- AIORESULT AIOUSB_GetMiscClock (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetCommTimeout (unsigned long DeviceIndex, unsigned timeout)
- unsigned AIOUSB_GetCommTimeout (unsigned long DeviceIndex)
- const char * AIOUSB_GetVersion (void)
- const char * AIOUSB_GetVersionDate (void)
- const char * AIOUSB_GetResultCodeAsString (unsigned long value)
- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- unsigned long AIOUSB_ADC_LoadCalTable (unsigned long DeviceIndex, const char *fileName)
- unsigned long AIOUSB_ADC_SetCalTable (unsigned long DeviceIndex, const unsigned short calTable[])
- unsigned long AIOUSB_ClearFIFO (unsigned long DeviceIndex, FIFO_Method Method)
- long AIOUSB_GetStreamingBlockSize (unsigned long DeviceIndex)

  This function is deprecated.
- AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock *config, unsigned long DeviceIndex, AIOUSB_BOOL defaults)
- AIORESULT GenericVendorRead (unsigned long deviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void *bufData, unsigned long *bytes_read)

  Performs basic low level USB vendor request.
- AIORESULT GenericVendorWrite (unsigned long DeviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void *bufData, unsigned long *bytes_write)

  Performs a generic vendor USB write.
- AIORESULT AIOUSB_Validate_Device (unsigned long DeviceIndex)

Variables

- int aio_errno
- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- unsigned long AIOUSB_INIT_PATTERN
- unsigned long aiousbInit

34.151.1 Macro Definition Documentation

#define PUBLIC_EXTERN extern

#define PRIVATE

#define PROD_NAME_SIZE 40

34.151.2 Typedef Documentation

typedef struct aiousboption AIOOption

typedef struct aioerror AIOError

34.151.3 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 ( )

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

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

build index of result codes

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex,* const char * *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex,* FIFO_Method *Method*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock * *config,* unsigned long *DeviceIndex,* AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

AIORESULT GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_read*)

Performs basic low level USB vendor request.

Returns

AIORESULT GenericVendorWrite (unsigned long *DeviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_write*)

Performs a generic vendor USB write.

AIORESULT AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

34.151.4 Variable Documentation

int aio_errno

struct ADRange adRanges[AD_NUM_GAIN_CODES]

unsigned long AIOUSB_INIT_PATTERN

unsigned long aiousbInit

34.152 lib/ndkbuild/jni/AIOUSB_Core.h File Reference

```
#include "AIOUSBDevice.h"
#include "libusb.h"
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
```

Data Structures

- struct BulkAcquireWorkerParams
- struct aiousboption
- struct aioerror
- struct ProductIDName
- struct ADRange

Macros

- #define PUBLIC_EXTERN extern
- #define PRIVATE
- #define PROD_NAME_SIZE 40

Typedefs

- typedef struct aiousboption AIOOption
- typedef struct aioerror AIOError

Functions

- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_ResetChip (unsigned long DeviceIndex)
- AIOUSB_BOOL AIOUSB_Lock (void)

  Notes on mutual exclusion / threading:
- AIOUSB_BOOL AIOUSB_UnLock (void)
- AIORESULT AIOUSB_InitTest (void)
- AIORESULT AIOUSB_Validate (unsigned long *DeviceIndex)
- AIORESULT AIOUSB_Validate_Lock (unsigned long *DeviceIndex)
- DeviceDescriptor * DeviceTableAtIndex (unsigned long DeviceIndex)
- DeviceDescriptor * DeviceTableAtIndex_Lock (unsigned long DeviceIndex)
- DeviceDescriptor * AIOUSB_GetDevice (unsigned long DeviceIndex)
- ADConfigBlock * AIOUSB_GetConfigBlock (DeviceDescriptor *dev)
- AIORESULT AIOUSB_SetMiscClock (unsigned long DeviceIndex, double clockHz)
- AIORESULT AIOUSB_GetMiscClock (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetCommTimeout (unsigned long DeviceIndex, unsigned timeout)
- unsigned AIOUSB_GetCommTimeout (unsigned long DeviceIndex)
- const char * AIOUSB_GetVersion (void)
- const char * AIOUSB_GetVersionDate (void)
- const char * AIOUSB_GetResultCodeAsString (unsigned long value)

- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- unsigned long AIOUSB_ADC_LoadCalTable (unsigned long DeviceIndex, const char *fileName)
- unsigned long AIOUSB_ADC_SetCalTable (unsigned long DeviceIndex, const unsigned short calTable[])
- unsigned long AIOUSB_ClearFIFO (unsigned long DeviceIndex, FIFO_Method Method)
- long AIOUSB_GetStreamingBlockSize (unsigned long DeviceIndex)

  This function is deprecated.
- AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock *config, unsigned long DeviceIndex, AIOUSB_BOOL defaults)
- AIORESULT GenericVendorRead (unsigned long deviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void *bufData, unsigned long *bytes_read)

  Performs basic low level USB vendor request.
- AIORESULT GenericVendorWrite (unsigned long DeviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void *bufData, unsigned long *bytes_write)

  Performs a generic vendor USB write.
- AIORESULT AIOUSB_Validate_Device (unsigned long DeviceIndex)

Variables

- int aio_errno
- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- unsigned long AIOUSB_INIT_PATTERN
- unsigned long aiousbInit

34.152.1 Macro Definition Documentation

#define PUBLIC_EXTERN extern

#define PRIVATE

#define PROD_NAME_SIZE 40

34.152.2 Typedef Documentation

typedef struct aiousboption AIOOption

typedef struct aioerror AIOError

34.152.3 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 ( )

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

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

build index of result codes

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex,* const char * *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex,* FIFO_Method *Method*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock ∗ *config,* unsigned long *DeviceIndex,* AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

AIORESULT GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void ∗ *bufData,* unsigned long ∗ *bytes_read*)

Performs basic low level USB vendor request.

Returns

AIORESULT GenericVendorWrite (unsigned long *DeviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void ∗ *bufData,* unsigned long ∗ *bytes_write*)

Performs a generic vendor USB write.

AIORESULT AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

34.152.4 Variable Documentation

int aio_errno

struct ADRange adRanges[AD_NUM_GAIN_CODES]

unsigned long AIOUSB_INIT_PATTERN

unsigned long aiousbInit

34.153 lib/AIOUSB_CTR.c File Reference

```
#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)

34.153.1 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 )
```

34.153.2 Function Documentation

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue,* unsigned char ∗ *pStatus*)

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double ∗ *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

34.154 lib/ndkbuild/jni/AIOUSB_CTR.c File Reference

```
#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)

34.154.1 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 )
```

34.154.2 Function Documentation

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue,* unsigned char ∗ *pStatus*)

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double ∗ *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

34.155 lib/AIOUSB_CTR.h File Reference

```
#include "AIOTypes.h"
#include "AIOUSB_Core.h"
```

Functions

- AIORET_TYPE CTR_CalculateCountersForClock (int hz, int ∗diva, int ∗divb)

  Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.
- AIORET_TYPE CTR_8254Mode (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode)
- AIORET_TYPE CTR_8254Load (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ReadModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254Read (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254ReadAll (unsigned long DeviceIndex, unsigned short ∗pData)
- AIORET_TYPE CTR_8254ReadStatus (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short ∗pReadValue, unsigned char ∗pStatus)
- AIORET_TYPE CTR_StartOutputFreq (unsigned long DeviceIndex, unsigned long BlockIndex, double ∗pHz)
- AIORET_TYPE CTR_8254SelectGate (unsigned long DeviceIndex, unsigned long GateIndex)
- AIORET_TYPE CTR_8254ReadLatched (unsigned long DeviceIndex, unsigned short ∗pData)

34.155.1 Function Documentation

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short * *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short * *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short * *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short * *pReadValue,* unsigned char * *pStatus*)

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double * *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short * *pData*)

34.156 lib/ndkbuild/jni/AIOUSB_CTR.h File Reference

```
#include "AIOTypes.h"
#include "AIOUSB_Core.h"
```

Functions

- AIORET_TYPE CTR_CalculateCountersForClock (int hz, int *diva, int *divb)

  *Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz *.*
- AIORET_TYPE CTR_8254Mode (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode)
- AIORET_TYPE CTR_8254Load (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ReadModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue, unsigned short *pReadValue)
- AIORET_TYPE CTR_8254Read (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short *pReadValue)
- AIORET_TYPE CTR_8254ReadAll (unsigned long DeviceIndex, unsigned short *pData)
- AIORET_TYPE CTR_8254ReadStatus (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short *pReadValue, unsigned char *pStatus)
- AIORET_TYPE CTR_StartOutputFreq (unsigned long DeviceIndex, unsigned long BlockIndex, double *pHz)
- AIORET_TYPE CTR_8254SelectGate (unsigned long DeviceIndex, unsigned long GateIndex)
- AIORET_TYPE CTR_8254ReadLatched (unsigned long DeviceIndex, unsigned short *pData)

34.156.1 Function Documentation

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int * *diva,* int * *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz *.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue,* unsigned char ∗ *pStatus*)

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double ∗ *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

34.157 lib/AIOUSB_CustomEEPROM.c File Reference

```
#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)

34.157.1 Macro Definition Documentation

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                          \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;              \
            goto g;                                          \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {         \
            goto g;                                          \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {  \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;             \
            goto g;                                          \
        }                                                    \
    } while (0 )
```

34.157.2 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)

34.158 lib/ndkbuild/jni/AIOUSB_CustomEEPROM.c File Reference

```
#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)

34.158.1 Macro Definition Documentation

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                             \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;                 \
            goto g;                                             \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {            \
            goto g;                                             \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {    \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;                \
            goto g;                                             \
        }                                                       \
    } while (0 )
```

34.158.2 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)

34.159 lib/AIOUSB_CustomEEPROM.h File Reference

```
#include "AIOTypes.h"
```

Functions

- unsigned long CustomEEPROMWrite (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long DataSize, void *Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)
- unsigned long CustomEEPROMRead (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long *DataSize, void *Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

34.159.1 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)

34.160 lib/ndkbuild/jni/AIOUSB_CustomEEPROM.h File Reference

```
#include "AIOTypes.h"
```

Functions

- unsigned long CustomEEPROMWrite (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long DataSize, void *Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)
- unsigned long CustomEEPROMRead (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long *DataSize, void *Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

34.160.1 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)

34.161 lib/AIOUSB_DAC.c File Reference

```
#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)

34.161.1 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*)

34.162 lib/ndkbuild/jni/AIOUSB_DAC.c File Reference

```
#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)

34.162.1 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*)

34.163 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)

34.163.1 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

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*)

34.164 lib/ndkbuild/jni/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)

34.164.1 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

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*)

34.165 lib/AIOUSB_DIO.c File Reference

```
#include "AIOUSB_DIO.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include "USBDevice.h"
#include <arpa/inet.h>
```

Macros

- #define GET_ENDPOINT(isread) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

Functions

- int MASK_BYTES_SIZE (AIOUSBDevice ∗device)
- int TRISTATE_BYTES_SIZE (AIOUSBDevice ∗device)
- unsigned short aiousb_htons (unsigned short octaveOffset)

  Returns the number in Big-Endian format.
- AIORESULT DIO_ConfigureWithDIOBuf (unsigned long DeviceIndex, unsigned char bTristate, AIOChannelMask ∗mask, DIOBuf ∗buf)
- AIORESULT DIO_Configure (unsigned long DeviceIndex, unsigned char bTristate, void ∗pOutMask, void ∗pData)
- AIORESULT DIO_ConfigureEx (unsigned long DeviceIndex, void ∗pOutMask, void ∗pData, void ∗pTristateMask)
- AIORESULT DIO_ConfigurationQuery (unsigned long DeviceIndex, void ∗pOutMask, void ∗pTristateMask)
- AIORESULT DIO_WriteAll (unsigned long DeviceIndex, void ∗pData)
- AIORESULT DIO_Write8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char Data)
- AIORESULT DIO_Write1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char bData)
- AIORESULT DIO_ReadAll (unsigned long DeviceIndex, void ∗buf)
- AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf)
- AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf)
- AIORESULT DIO_ReadAllToCharStr (unsigned long DeviceIndex, char ∗buf, unsigned size)
- AIORESULT DIO_Read8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char ∗pdat)
- AIORESULT DIO_Read1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char ∗bit)
- AIORESULT DIO_StreamOpen (unsigned long DeviceIndex, unsigned long bIsRead)
- AIORESULT DIO_StreamClose (unsigned long DeviceIndex)
- AIORESULT DIO_StreamSetClocks (unsigned long DeviceIndex, double ∗ReadClockHz, double ∗WriteClockHz)
- int pow_of_minsize (int val)
- AIORESULT DIO_StreamFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗pFrameData, unsigned long ∗BytesTransferred)

34.165.1 Macro Definition Documentation

#define GET_ENDPOINT(*isread*) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

34.165.2 Function Documentation

int MASK_BYTES_SIZE (AIOUSBDevice * *device*)

int TRISTATE_BYTES_SIZE (AIOUSBDevice * *device*)

unsigned short aiousb_htons (unsigned short *octaveOffset*)

Returns the number in Big-Endian format.

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask * *mask,* DIOBuf * *buf*)

AIORESULT DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void * *pOutMask,* void * *pData*)

AIORESULT DIO_ConfigureEx (unsigned long *DeviceIndex,* void * *pOutMask,* void * *pData,* void * *pTristateMask*)

AIORESULT DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void * *pOutMask,* void * *pTristateMask*)

AIORESULT DIO_WriteAll (unsigned long *DeviceIndex,* void * *pData*)

AIORESULT DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

AIORESULT DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* void * *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

AIORESULT DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char * *buf,* unsigned *size*)

AIORESULT DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char * *pdat*)

AIORESULT DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char * *bit*)

AIORESULT DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

AIORESULT DIO_StreamClose (unsigned long *DeviceIndex*)

AIORESULT DIO_StreamSetClocks (unsigned long *DeviceIndex,* double * *ReadClockHz,* double * *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```

int pow_of_minsize (int *val*)

AIORESULT DIO_StreamFrame (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short * *pFrameData,* unsigned long * *BytesTransferred*)

Note

convert parameter types to those that libusb likes

34.166 lib/ndkbuild/jni/AIOUSB_DIO.c File Reference

```
#include "AIOUSB_DIO.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include "USBDevice.h"
#include <arpa/inet.h>
```

Macros

- #define GET_ENDPOINT(isread) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

Functions

- int MASK_BYTES_SIZE (AIOUSBDevice *device)
- int TRISTATE_BYTES_SIZE (AIOUSBDevice *device)
- unsigned short aiousb_htons (unsigned short octaveOffset)

  Returns the number in Big-Endian format.
- AIORESULT DIO_ConfigureWithDIOBuf (unsigned long DeviceIndex, unsigned char bTristate, AIOChannelMask *mask, DIOBuf *buf)
- AIORESULT DIO_Configure (unsigned long DeviceIndex, unsigned char bTristate, void *pOutMask, void *pData)
- AIORESULT DIO_ConfigureEx (unsigned long DeviceIndex, void *pOutMask, void *pData, void *pTristateMask)
- AIORESULT DIO_ConfigurationQuery (unsigned long DeviceIndex, void *pOutMask, void *pTristateMask)
- AIORESULT DIO_WriteAll (unsigned long DeviceIndex, void *pData)
- AIORESULT DIO_Write8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char Data)
- AIORESULT DIO_Write1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char bData)
- AIORESULT DIO_ReadAll (unsigned long DeviceIndex, void *buf)
- AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf *buf)
- AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long DeviceIndex, DIOBuf *buf)
- AIORESULT DIO_ReadAllToCharStr (unsigned long DeviceIndex, char *buf, unsigned size)
- AIORESULT DIO_Read8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char *pdat)
- AIORESULT DIO_Read1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char *bit)
- AIORESULT DIO_StreamOpen (unsigned long DeviceIndex, unsigned long bIsRead)
- AIORESULT DIO_StreamClose (unsigned long DeviceIndex)
- AIORESULT DIO_StreamSetClocks (unsigned long DeviceIndex, double *ReadClockHz, double *WriteClockHz)
- int pow_of_minsize (int val)
- AIORESULT DIO_StreamFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short *pFrameData, unsigned long *BytesTransferred)

34.166.1 Macro Definition Documentation

#define GET_ENDPOINT(*isread*) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

34.166.2 Function Documentation

int MASK_BYTES_SIZE (AIOUSBDevice * *device*)

int TRISTATE_BYTES_SIZE (AIOUSBDevice * *device*)

unsigned short aiousb_htons (unsigned short *octaveOffset*)

Returns the number in Big-Endian format.

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* **AIOChannelMask** ∗ *mask,* **DIOBuf** ∗ *buf*)

AIORESULT DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* **void** ∗ *pOutMask,* **void** ∗ *pData*)

AIORESULT DIO_ConfigureEx (unsigned long *DeviceIndex,* **void** ∗ *pOutMask,* **void** ∗ *pData,* **void** ∗ *pTristateMask*)

AIORESULT DIO_ConfigurationQuery (unsigned long *DeviceIndex,* **void** ∗ *pOutMask,* **void** ∗ *pTristateMask*)

AIORESULT DIO_WriteAll (unsigned long *DeviceIndex,* **void** ∗ *pData*)

AIORESULT DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

AIORESULT DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* **void** ∗ *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* **DIOBuf** ∗ *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* **DIOBuf** ∗ *buf*)

AIORESULT DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char ∗ *buf,* unsigned *size*)

AIORESULT DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char ∗ *pdat*)

AIORESULT DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char ∗ *bit*)

AIORESULT DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

AIORESULT DIO_StreamClose (unsigned long *DeviceIndex*)

AIORESULT DIO_StreamSetClocks (unsigned long *DeviceIndex,* double ∗ *ReadClockHz,* double ∗ *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```

int pow_of_minsize (int *val*)

AIORESULT DIO_StreamFrame (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short ∗ *pFrameData,* unsigned long ∗ *BytesTransferred*)

Note

convert parameter types to those that libusb likes

34.167 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 bIsRead)
- 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)

34.167.1 Function Documentation

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask ∗ *mask,* DIOBuf ∗ *buf*)

unsigned long DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void ∗ *pOutMask,* void ∗ *pData*)

unsigned long DIO_ConfigureEx (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pData,* void ∗ *pTristateMask*)

unsigned long DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pTristateMask*)

unsigned long DIO_WriteAll (unsigned long *DeviceIndex,* void ∗ *pData*)

unsigned long DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

unsigned long DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex*, void * *buf*)

unsigned long DIO_ReadAllToCharStr (unsigned long *DeviceIndex*, char * *buf*, unsigned *size*)

unsigned long DIO_Read8 (unsigned long *DeviceIndex*, unsigned long *ByteIndex*, unsigned char * *pdat*)

unsigned long DIO_Read1 (unsigned long *DeviceIndex*, unsigned long *BitIndex*, unsigned char * *bit*)

unsigned long DIO_StreamOpen (unsigned long *DeviceIndex*, unsigned long *bIsRead*)

unsigned long DIO_StreamClose (unsigned long *DeviceIndex*)

unsigned long DIO_StreamSetClocks (unsigned long *DeviceIndex*, double * *ReadClockHz*, double * *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```


```
* 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
convert parameter types to those that libusb likes

34.168 lib/ndkbuild/jni/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 bIsRead)
- 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)

34.168.1 Function Documentation

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask * *mask,* DIOBuf * *buf*)

unsigned long DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void * *pOutMask,* void * *pData*)

unsigned long DIO_ConfigureEx (unsigned long *DeviceIndex,* void * *pOutMask,* void * *pData,* void * *pTristateMask*)

unsigned long DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void * *pOutMask,* void * *pTristateMask*)

unsigned long DIO_WriteAll (unsigned long *DeviceIndex,* void * *pData*)

unsigned long DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

unsigned long DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* void * *buf*)

unsigned long DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char * *buf,* unsigned *size*)

unsigned long DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char * *pdat*)

unsigned long DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char * *bit*)

unsigned long DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

unsigned long DIO_StreamClose (unsigned long *DeviceIndex*)

unsigned long DIO_StreamSetClocks (unsigned long *DeviceIndex,* double * *ReadClockHz,* double * *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
* 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
convert parameter types to those that libusb likes

34.169 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

34.169.1 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

34.170 lib/ndkbuild/jni/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

34.170.1 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

34.171 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, 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

34.171.1 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__)

34.171.2 Enumeration Type Documentation

enum AIO_DEBUG_LEVEL

Enumerator

AIOFATAL_LEVEL

AIOERROR_LEVEL

AIOINFO_LEVEL

AIOWARN_LEVEL

AIODEFAULT_LOG_LEVEL

AIODEBUG_LEVEL

AIODEVEL_LEVEL

AIOFATAL_LEVEL

AIOERROR_LEVEL

AIOINFO_LEVEL

AIOWARN_LEVEL

AIODEFAULT_LOG_LEVEL

AIODEBUG_LEVEL

AIODEVEL_LEVEL

34.171.3 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

34.172 lib/ndkbuild/jni/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, 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

34.172.1 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__)

34.172.2 Enumeration Type Documentation

enum AIO_DEBUG_LEVEL

Enumerator

AIOFATAL_LEVEL

AIOERROR_LEVEL

AIOINFO_LEVEL

AIOWARN_LEVEL

AIODEFAULT_LOG_LEVEL

AIODEBUG_LEVEL

AIODEVEL_LEVEL

AIOFATAL_LEVEL

AIOERROR_LEVEL

AIOINFO_LEVEL

AIOWARN_LEVEL

AIODEFAULT_LOG_LEVEL

AIODEBUG_LEVEL

AIODEVEL_LEVEL

34.172.3 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

34.173 lib/AIOUSB_Properties.c File Reference

```
#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 ()

34.173.1 Macro Definition Documentation

#define RESULT_TEXT_SIZE 40

#define NUM_RESULT_CODES (sizeof(resultCodeTable) / sizeof(resultCodeTable[0]))

34.173.2 Function Documentation

int non_usb_supported_device (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

unsigned long AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices ∗ 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t ∗ *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

unsigned long GetDeviceBySerialNumber (uint64_t *serialNumber*)

else, even if we get an error requesting the serial number from this device, keep searching

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

unsigned long AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties ∗ *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_ListDevices ( )

34.174 lib/ndkbuild/jni/AIOUSB_Properties.c File Reference

```
#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 ()

34.174.1 Macro Definition Documentation

#define RESULT_TEXT_SIZE 40

#define NUM_RESULT_CODES (sizeof(resultCodeTable) / sizeof(resultCodeTable[0]))

34.174.2 Function Documentation

int non_usb_supported_device (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

unsigned long AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices ∗ 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t ∗ *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

unsigned long GetDeviceBySerialNumber (uint64_t *serialNumber*)

else, even if we get an error requesting the serial number from this device, keep searching

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

unsigned long AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties ∗ *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_ListDevices ( )

34.175 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,
  BASIC = 0, TERSE = 1, JSON = 2, YAML = 3 }

Functions

- AIORESULT AIOUSB_GetDeviceByProductID (int minProductID, int maxProductID, int maxDevices, int *deviceList)
- AIORESULT GetDeviceBySerialNumber (uint64_t pSerialNumber)
- AIORESULT GetDeviceSerialNumber (unsigned long DeviceIndex, uint64_t *pSerialNumber)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORESULT AIOUSB_GetDeviceProperties (unsigned long DeviceIndex, DeviceProperties *properties)

  AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()
- const char * AIOUSB_GetResultCodeAsString (unsigned long result_value)
- AIORET_TYPE AIOUSB_ListDevices ()
- AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType display_type)
- AIORET_TYPE AIOUSB_FindDevices (int **where, int *length, AIOUSB_BOOL(*is_ok_device)(AIOUSBDevice *dev))

  Friendly function that can be called first.
- AIORET_TYPE AIOUSB_FindDevicesByGroup (int **where, int *length, AIOProductGroup *pg)
- AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist *indices, AIOProductGroup *pg)

34.175.1 Enumeration Type Documentation

enum AIODisplayType

Enumerator

BASIC

TERSE

JSON

YAML

BASIC

TERSE

JSON

YAML

34.175.2 Function Documentation

AIORESULT AIOUSB_GetDeviceByProductID (int *minProductID*, int *maxProductID*, int *maxDevices*, int * *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices * 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

< 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

else, even if we get an error requesting the serial number from this device, keep searching

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex*, uint64_t * *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties ∗ *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

build index of result codes

AIORET_TYPE AIOUSB_ListDevices ( )

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

34.176 lib/ndkbuild/jni/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,
  BASIC = 0, TERSE = 1, JSON = 2, YAML = 3 }

Functions

- AIORESULT AIOUSB_GetDeviceByProductID (int minProductID, int maxProductID, int maxDevices, int ∗deviceList)
- AIORESULT GetDeviceBySerialNumber (uint64_t pSerialNumber)
- AIORESULT GetDeviceSerialNumber (unsigned long DeviceIndex, uint64_t ∗pSerialNumber)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORESULT AIOUSB_GetDeviceProperties (unsigned long DeviceIndex, DeviceProperties ∗properties)

  AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()
- const char ∗ AIOUSB_GetResultCodeAsString (unsigned long result_value)
- AIORET_TYPE AIOUSB_ListDevices ()
- AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType display_type)
- AIORET_TYPE AIOUSB_FindDevices (int ∗∗where, int ∗length, AIOUSB_BOOL(∗is_ok_device)(AIOUSBDevice ∗dev))

Friendly function that can be called first.

- AIORET_TYPE AIOUSB_FindDevicesByGroup (int **where, int *length, AIOProductGroup *pg)
- AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist *indices, AIOProductGroup *pg)

34.176.1 Enumeration Type Documentation

enum AIODisplayType

Enumerator

BASIC

TERSE

JSON

YAML

BASIC

TERSE

JSON

YAML

34.176.2 Function Documentation

AIORESULT AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int * *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices * 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

< 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

else, even if we get an error requesting the serial number from this device, keep searching

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t * *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties * *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char* AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

build index of result codes

AIORET_TYPE AIOUSB_ListDevices ( )

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

34.177 lib/AIOUSB_USB.c File Reference

```
#include "AIOUSB_USB.h"
```

34.178 lib/ndkbuild/jni/AIOUSB_USB.c File Reference

```
#include "AIOUSB_USB.h"
```

34.179 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

34.179.1 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

34.180 lib/ndkbuild/jni/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

34.180.1 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

34.181 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.

34.181.1 Function Documentation

AIOWDGConfig* NewWDGConfig (void)

Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.

Returns

AIOWDGConfig *obj New Watchdog configuration object

void doSomething ( )

void DeleteWDGConfig (AIOWDGConfig * *obj*)

Deletes the AIOWDGConfig object.

Parameters

obj	

AIORET_TYPE WDG_SetConfig (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Assigns the watchdog object to the device index in question.

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_GetStatus (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_Pet (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Pets the watchdog and keeps it from resetting the device.

Parameters

DeviceIndex	
obj	

Returns

>= 0 if successful, < 0 otherwise

34.182 lib/ndkbuild/jni/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.

34.182.1 Function Documentation

AIOWDGConfig* NewWDGConfig (void)

Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.

Returns

AIOWDGConfig *obj New Watchdog configuration object

void doSomething ( )

void DeleteWDGConfig (AIOWDGConfig * *obj*)

Deletes the AIOWDGConfig object.

Parameters

obj	

AIORET_TYPE WDG_SetConfig (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Assigns the watchdog object to the device index in question.

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_GetStatus (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_Pet (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Pets the watchdog and keeps it from resetting the device.

Parameters

DeviceIndex	
obj	

Returns

>= 0 if successful, < 0 otherwise

34.183 lib/AIOUSB_WDG.h File Reference

```
#include "aiousb.h"
```

Data Structures

- struct AIOWDGConfig

Enumerations

- enum WDGVals {
  WDGVals_begin = (0x00 -1), AIOUSB_WDG_READ_VALUE, AIOUSB_WDG_READ_INDEX = 0x0041, WDG-Vals_end,
  WDGVals_begin = (0x00 -1), AIOUSB_WDG_READ_VALUE, AIOUSB_WDG_READ_INDEX = 0x0041, WDG-Vals_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.

34.183.1 Enumeration Type Documentation

enum WDGVals

Enumerator

WDGVals_begin

AIOUSB_WDG_READ_VALUE

AIOUSB_WDG_READ_INDEX

WDGVals_end

WDGVals_begin

AIOUSB_WDG_READ_VALUE

AIOUSB_WDG_READ_INDEX

WDGVals_end

34.183.2 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

34.184 lib/ndkbuild/jni/AIOUSB_WDG.h File Reference

```
#include "aiousb.h"
```

Data Structures

- struct AIOWDGConfig

Enumerations

- enum WDGVals {
  WDGVals_begin = (0x00 -1), AIOUSB_WDG_READ_VALUE, AIOUSB_WDG_READ_INDEX = 0x0041, WDGVals_end,
  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.

34.184.1 Enumeration Type Documentation

enum WDGVals

Enumerator

WDGVals_begin
AIOUSB_WDG_READ_VALUE
AIOUSB_WDG_READ_INDEX
WDGVals_end
WDGVals_begin
AIOUSB_WDG_READ_VALUE
AIOUSB_WDG_READ_INDEX
WDGVals_end

34.184.2 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

34.185 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)

34.185.1 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*)

34.186 lib/ndkbuild/jni/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)

34.186.1 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*)

34.187 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)

34.187.1 Typedef Documentation

typedef **AIOUSBDevice DeviceDescriptor**

34.187.2 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*)

34.188 lib/ndkbuild/jni/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)

34.188.1 Typedef Documentation

typedef AIOUSBDevice DeviceDescriptor

34.188.2 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*)

34.189 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)

34.189.1 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*)

34.190 lib/ndkbuild/jni/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)

34.190.1 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*)

34.191 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)

34.191.1 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))

34.191.2 Typedef Documentation

typedef struct cJSON cJSON

typedef struct cJSON_Hooks cJSON_Hooks

34.191.3 Function Documentation

void cJSON_InitHooks (cJSON_Hooks * *hooks*)

cJSON* cJSON_Parse (const char * *value*)

char* cJSON_Print (cJSON * *item*)

char* cJSON_PrintUnformatted (cJSON * *item*)

void cJSON_Delete (cJSON * *c*)

int cJSON_AsInteger (cJSON * *item*)

int cJSON_GetArraySize (cJSON * *array*)

cJSON* cJSON_GetArrayItem (cJSON * *array,* int *item*)

cJSON* cJSON_GetObjectItem (cJSON * *object,* const char * *string*)

const char* cJSON_GetErrorPtr (void)

cJSON* cJSON_CreateNull (void)

cJSON* cJSON_CreateTrue (void)

cJSON* cJSON_CreateFalse (void)

cJSON* cJSON_CreateBool (int *b*)

cJSON* cJSON_CreateNumber (double *num*)

cJSON* cJSON_CreateString (const char * *string*)

cJSON* cJSON_CreateArray (void)

cJSON* cJSON_CreateObject (void)

cJSON* cJSON_CreateIntArray (const int * *numbers,* int *count*)

cJSON* cJSON_CreateFloatArray (const float * *numbers,* int *count*)

cJSON* cJSON_CreateDoubleArray (const double * *numbers,* int *count*)

cJSON* cJSON_CreateStringArray (const char ** *strings,* int *count*)

void cJSON_AddItemToArray (cJSON * *array,* cJSON * *item*)

void cJSON_AddItemToObject (cJSON * *object,* const char * *string,* cJSON * *item*)

void cJSON_AddItemReferenceToArray (cJSON * *array,* cJSON * *item*)

void cJSON_AddItemReferenceToObject (cJSON * *object,* const char * *string,* cJSON * *item*)

cJSON* cJSON_DetachItemFromArray (cJSON * *array,* int *which*)

void cJSON_DeleteItemFromArray (cJSON * *array,* int *which*)

cJSON* cJSON_DetachItemFromObject (cJSON * *object,* const char * *string*)

void cJSON_DeleteItemFromObject (cJSON * *object,* const char * *string*)

void cJSON_ReplaceItemInArray (cJSON * *array,* int *which,* cJSON * *newitem*)

void cJSON_ReplaceItemInObject (cJSON * *object,* const char * *string,* cJSON * *newitem*)

cJSON* cJSON_Duplicate (cJSON * *item,* int *recurse*)

cJSON* cJSON_ParseWithOpts (const char * *value,* const char ** *return_parse_end,* int *require_null_terminated*)

void cJSON_Minify (char * *json*)

34.192 lib/ndkbuild/jni/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)

34.192.1 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))

34.192.2 Typedef Documentation

typedef struct **cJSON** **cJSON**

typedef struct **cJSON_Hooks** **cJSON_Hooks**

34.192.3 Function Documentation

void cJSON_InitHooks (**cJSON_Hooks** ∗ *hooks*)

cJSON∗ cJSON_Parse (**const** char ∗ *value*)

char∗ cJSON_Print (**cJSON** ∗ *item*)

char∗ cJSON_PrintUnformatted (**cJSON** ∗ *item*)

void cJSON_Delete (**cJSON** ∗ *c*)

int cJSON_AsInteger (**cJSON** ∗ *item*)

int cJSON_GetArraySize (**cJSON** ∗ *array*)

cJSON∗ cJSON_GetArrayItem (**cJSON** ∗ *array,* int *item*)

cJSON∗ cJSON_GetObjectItem (**cJSON** ∗ *object,* **const** char ∗ *string*)

const char∗ cJSON_GetErrorPtr (**void**)

cJSON∗ cJSON_CreateNull (**void**)

cJSON∗ cJSON_CreateTrue (**void**)

cJSON∗ cJSON_CreateFalse (**void**)

cJSON∗ cJSON_CreateBool (int *b*)

cJSON∗ **cJSON_CreateNumber** (double *num*)

cJSON∗ **cJSON_CreateString** (const char ∗ *string*)

cJSON∗ **cJSON_CreateArray** (void)

cJSON∗ **cJSON_CreateObject** (void)

cJSON∗ **cJSON_CreateIntArray** (const int ∗ *numbers,* int *count*)

cJSON∗ **cJSON_CreateFloatArray** (const float ∗ *numbers,* int *count*)

cJSON∗ **cJSON_CreateDoubleArray** (const double ∗ *numbers,* int *count*)

cJSON∗ **cJSON_CreateStringArray** (const char ∗∗ *strings,* int *count*)

void cJSON_AddItemToArray (**cJSON** ∗ *array,* **cJSON** ∗ *item*)

void cJSON_AddItemToObject (**cJSON** ∗ *object,* const char ∗ *string,* **cJSON** ∗ *item*)

void cJSON_AddItemReferenceToArray (**cJSON** ∗ *array,* **cJSON** ∗ *item*)

void cJSON_AddItemReferenceToObject (**cJSON** ∗ *object,* const char ∗ *string,* **cJSON** ∗ *item*)

cJSON∗ **cJSON_DetachItemFromArray** (**cJSON** ∗ *array,* int *which*)

void cJSON_DeleteItemFromArray (**cJSON** ∗ *array,* int *which*)

cJSON∗ **cJSON_DetachItemFromObject** (**cJSON** ∗ *object,* const char ∗ *string*)

void cJSON_DeleteItemFromObject (**cJSON** ∗ *object,* const char ∗ *string*)

void cJSON_ReplaceItemInArray (**cJSON** ∗ *array,* int *which,* **cJSON** ∗ *newitem*)

void cJSON_ReplaceItemInObject (**cJSON** ∗ *object,* const char ∗ *string,* **cJSON** ∗ *newitem*)

cJSON∗ **cJSON_Duplicate** (**cJSON** ∗ *item,* int *recurse*)

cJSON∗ **cJSON_ParseWithOpts** (const char ∗ *value,* const char ∗∗ *return_parse_end,* int *require_null_terminated*)

void cJSON_Minify (char ∗ *json*)

34.193 lib/CStringArray.c File Reference

```
#include "CStringArray.h"
#include "AIOTypes.h"
#include <stdarg.h>
#include <string.h>
#include <stddef.h>
#include <stdio.h>
```

Functions

- CStringArray ∗ NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray ∗ NewCStringArray (size_t numstrings)
- CStringArray ∗ NewCStringArrayFromCArgs (int argc, char ∗argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray ∗str)
- CStringArray ∗ CopyCStringArray (CStringArray ∗str)
- char ∗ CStringArrayToString (CStringArray ∗str)
- char ∗ CStringArrayToStringWithDelimeter (CStringArray ∗str, const char ∗delim)

34.193.1 Function Documentation

CStringArray∗ **NewCStringArrayWithStrings** (size_t *numstrings,* ...)

CStringArray∗ **NewCStringArray** (size_t *numstrings*)

CStringArray* **NewCStringArrayFromCArgs** (int *argc,* char * *argv[]*)

AIORET_TYPE **DeleteCStringArray** (**CStringArray** * *str*)

CStringArray* **CopyCStringArray** (**CStringArray** * *str*)

char* **CStringArrayToString** (**CStringArray** * *str*)

char* **CStringArrayToStringWithDelimeter** (**CStringArray** * *str,* **const** char * *delim*)

34.194 lib/ndkbuild/jni/CStringArray.c File Reference

```
#include "CStringArray.h"
#include "AIOTypes.h"
#include <stdarg.h>
#include <string.h>
#include <stddef.h>
#include <stdio.h>
```

Functions

- CStringArray * NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray * NewCStringArray (size_t numstrings)
- CStringArray * NewCStringArrayFromCArgs (int argc, char *argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray *str)
- CStringArray * CopyCStringArray (CStringArray *str)
- char * CStringArrayToString (CStringArray *str)
- char * CStringArrayToStringWithDelimeter (CStringArray *str, const char *delim)

34.194.1 Function Documentation

CStringArray* **NewCStringArrayWithStrings** (size_t *numstrings,* ...)

CStringArray* **NewCStringArray** (size_t *numstrings*)

CStringArray* **NewCStringArrayFromCArgs** (int *argc,* char * *argv[]*)

AIORET_TYPE **DeleteCStringArray** (**CStringArray** * *str*)

CStringArray* **CopyCStringArray** (**CStringArray** * *str*)

char* **CStringArrayToString** (**CStringArray** * *str*)

char* **CStringArrayToStringWithDelimeter** (**CStringArray** * *str,* **const** char * *delim*)

34.195 lib/CStringArray.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct CStringArray

Macros

- #define STRING_ARRAY(N,...)

Typedefs

- typedef struct CStringArray CStringArray

Functions

- CStringArray * NewCStringArray (size_t numstrings)
- CStringArray * NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray * NewCStringArrayFromCArgs (int argc, char *argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray *str)
- CStringArray * CopyCStringArray (CStringArray *str)
- char * CStringArrayToString (CStringArray *str)
- char * CStringArrayToStringWithDelimeter (CStringArray *str, const char *delim)

34.195.1 Macro Definition Documentation

#define STRING_ARRAY(*N, ...*)

34.195.2 Typedef Documentation

typedef struct CStringArray CStringArray

34.195.3 Function Documentation

CStringArray* NewCStringArray (size_t *numstrings*)

CStringArray* NewCStringArrayWithStrings (size_t *numstrings, ...*)

CStringArray* NewCStringArrayFromCArgs (int *argc,* char * *argv[]*)

AIORET_TYPE DeleteCStringArray (CStringArray * *str*)

CStringArray* CopyCStringArray (CStringArray * *str*)

char* CStringArrayToString (CStringArray * *str*)

char* CStringArrayToStringWithDelimeter (CStringArray * *str,* const char * *delim*)

34.196 lib/ndkbuild/jni/CStringArray.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct CStringArray

Macros

- #define STRING_ARRAY(N,...)

Typedefs

- typedef struct CStringArray CStringArray

Functions

- CStringArray * NewCStringArray (size_t numstrings)
- CStringArray * NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray * NewCStringArrayFromCArgs (int argc, char *argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray *str)
- CStringArray * CopyCStringArray (CStringArray *str)
- char * CStringArrayToString (CStringArray *str)
- char * CStringArrayToStringWithDelimeter (CStringArray *str, const char *delim)

34.196.1 Macro Definition Documentation

#define STRING_ARRAY(*N, ...*)

34.196.2 Typedef Documentation

typedef struct CStringArray CStringArray

34.196.3 Function Documentation

CStringArray∗ NewCStringArray (size_t *numstrings*)

CStringArray∗ NewCStringArrayWithStrings (size_t *numstrings, ...*)

CStringArray∗ NewCStringArrayFromCArgs (int *argc,* char ∗ *argv[]*)

AIORET_TYPE DeleteCStringArray (CStringArray ∗ *str*)

CStringArray∗ CopyCStringArray (CStringArray ∗ *str*)

char∗ CStringArrayToString (CStringArray ∗ *str*)

char∗ CStringArrayToStringWithDelimeter (CStringArray ∗ *str,* const char ∗ *delim*)

34.197 lib/DIOBuf.c File Reference

```
#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)

34.197.1 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary,* int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf∗ NewDIOBufFromBinStr (const char ∗ *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

DIOBuf∗ DIOBufReplaceString (DIOBuf ∗ *buf,* char ∗ *ary,* int *size_array*)

Replaces the content of the buffer buf with the new array , of size size ∗.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf∗ DIOBufReplaceBinString (DIOBuf ∗ *buf,* char ∗ *bitstr*)

void DeleteDIOBuf (DIOBuf ∗ *buf*)

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf,* unsigned *newsize*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

char* DIOBufToHex (DIOBuf * *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char* DIOBufToBinary (DIOBuf * *buf*)

char* DIOBufToInvertedBinary (DIOBuf * *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

AIORET_TYPE DIOBufSetIndex (DIOBuf * *buf*, int *index*, unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf * *buf*, int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char ∗ *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char *value*)

34.198 lib/ndkbuild/jni/DIOBuf.c File Reference

```
#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)

34.198.1 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary,* int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf∗ NewDIOBufFromBinStr (const char ∗ *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

DIOBuf∗ DIOBufReplaceString (DIOBuf ∗ *buf,* char ∗ *ary,* int *size_array*)

Replaces the content of the buffer buf with the new array , of size size ∗.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf∗ DIOBufReplaceBinString (DIOBuf ∗ *buf,* char ∗ *bitstr*)

void DeleteDIOBuf (DIOBuf ∗ *buf*)

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf,* unsigned *newsize*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

char∗ DIOBufToHex (DIOBuf ∗ *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char* DIOBufToBinary (DIOBuf * *buf*)

char* DIOBufToInvertedBinary (DIOBuf * *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

AIORET_TYPE DIOBufSetIndex (DIOBuf * *buf*, int *index*, unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf * *buf*, int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf * *buf*, unsigned *index*, char * *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf * *buf*, unsigned *index*, char *value*)

34.199 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)

34.199.1 Typedef Documentation

typedef unsigned char DIOBufferType

34.199.2 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary*, int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf* **NewDIOBufFromBinStr** (**const** char * *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

void DeleteDIOBuf (DIOBuf ∗ *buf*)

DIOBuf∗ DIOBufReplaceString (DIOBuf ∗ *buf*, char ∗ *ary*, int *size_array*)

Replaces the content of the buffer buf with the new array , of size size ∗.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf∗ DIOBufReplaceBinString (DIOBuf ∗ *buf*, char ∗ *bitstr*)

char∗ DIOBufToHex (DIOBuf ∗ *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char∗ DIOBufToBinary (DIOBuf ∗ *buf*)

char∗ DIOBufToInvertedBinary (DIOBuf ∗ *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf*, unsigned *size*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

AIORET_TYPE DIOBufSetIndex (DIOBuf * *buf*, int *index*, unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf * *buf*, int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf * *buf*, unsigned *index*, char * *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf * *buf*, unsigned *index*, char *value*)

34.200 lib/ndkbuild/jni/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)

34.200.1 Typedef Documentation

typedef unsigned char DIOBufferType

34.200.2 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary*, int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf∗ NewDIOBufFromBinStr (const char ∗ *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

void DeleteDIOBuf (DIOBuf * *buf*)

DIOBuf* DIOBufReplaceString (DIOBuf * *buf*, char * *ary*, int *size_array*)

Replaces the content of the buffer buf with the new array , of size size *.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf* DIOBufReplaceBinString (DIOBuf * *buf*, char * *bitstr*)

char* DIOBufToHex (DIOBuf * *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char* DIOBufToBinary (DIOBuf * *buf*)

char* DIOBufToInvertedBinary (DIOBuf * *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

DIOBuf* DIOBufResize (DIOBuf * *buf*, unsigned *size*)

unsigned DIOBufSize (DIOBuf * *buf*)

unsigned DIOBufByteSize (DIOBuf * *buf*)

char* DIOBufToString (DIOBuf * *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

AIORET_TYPE DIOBufSetIndex (DIOBuf ∗ *buf,* int *index,* unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf ∗ *buf,* int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char ∗ *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char *value*)

34.201 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)

34.201.1 Function Documentation

AIORET_TYPE aiocontbuf_get_bulk_data (AIOContinuousBuf ∗ *buf,* USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

34.202 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)

34.202.1 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)

34.203 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

34.203.1 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

34.203.2 Typedef Documentation

typedef AIOEither(* init_device)(USBDevice *usb, LIBUSBArgs *args)

34.203.3 Enumeration Type Documentation

enum IO_DIRECTION

Enumerator

IN
OUT
IN
OUT
IN
OUT

34.203.4 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.

34.203.5 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

34.204 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)

34.204.1 Enumeration Type Documentation

enum IO_DIRECTION

Enumerator

IN
OUT
IN
OUT
IN
OUT

34.204.2 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*)

34.205 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

34.205.1 Typedef Documentation

typedef **AIORET_TYPE**(* add_devices_fn)(**libusb_device** **deviceList, **USBDevice** **devs, int *size)

34.205.2 Enumeration Type Documentation

enum **IO_DIRECTION**

Enumerator

IN

OUT

IN

OUT

IN

OUT

34.205.3 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*)

34.205.4 Variable Documentation

IO_DIRECTION direction

ADCConfigBlock* **KEEP = NULL**

34.206 lib/ndkbuild/jni/libaiousb.h File Reference

```
#include <stdint.h>
#include "AIOUSB_Core.h"
#include "AIOChannelMask.h"
#include "AIOContinuousBuffer.h"
#include "AIOTypes.h"
#include "DIOBuf.h"
#include "AIODeviceInfo.h"
#include "AIODeviceQuery.h"
#include "AIOCommandLine.h"
#include "AIOUSB_Properties.h"
#include <mex.h>
```

Data Structures

- struct ad_gain_pairs
- struct aio_channel_range
- struct AIOGainRange
- struct aio_counts_converter
- struct AIOWDGConfig

Macros

- #define MATLAB
- #define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain_pairs) - 1))

Typedefs

- typedef unsigned long AIORESULT
- typedef int64_t AIORET_TYPE
- typedef struct aio_channel_range AIOChannelRange
- typedef struct aio_counts_converter AIOCountsConverter

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)
- 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 bIsRead)
- 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)
- 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)
- AIOContinuousBuf * NewAIOContinuousBuf (unsigned long DeviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIOContinuousBuf * NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf * NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf *buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf *buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf *buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf *buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf *buf, void *(*work)(void *object))
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf *buf, unsigned sblksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf *buf)
- ADCConfigBlock * AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf *buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf *buf, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf *buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf *buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf *buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf *buf)

- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf *buf, ADGainCode gain, AIOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf *buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf *buf, AIOChannelMask *mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf *buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf *buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf *buf, unsigned timeout)
- AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf *buf, AIOUSB_BOOL debug)
- AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf *buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf *buf, void *frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf *buf, void *tobuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf *buf)

  Setups the Automated runs for continuous mode runs.
- AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf *buf, AIOCmd *cmd, AIORET_TYPE(*callback)(AIOContinuousBuf *buf))

  *Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.*
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf *buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf *buf, unsigned short *tmp, unsigned tmpsize, unsigned size)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned read_buf_size)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf *buf, unsigned short *read_buf, unsigned tmpbuffer_size, int64_t num_scans)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf *buf)

  returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf *buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf *buf)
- AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf *buf, ADGainCode gainCode)
- AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf *buf, AIOBufferType *readbuf, unsigned readbufsize, unsigned size)

  *Reads the current available amount of data from buf, into the readbuf datastructure *.*
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf *buf, AIOBufferType *writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf *buf, unsigned short *data, unsigned datasize, unsigned size, AIOContinuousBufMode flag)
- AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf *buf)
- char * AIOContinuousBufToJSON (AIOContinuousBuf *buf)
- AIOContinuousBuf * NewAIOContinuousBufFromJSON (const char *json_string)
- AIORET_TYPE CTR_CalculateCountersForClock (int hz, int *diva, int *divb)

  *Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz *.*

- AIORET_TYPE CTR_8254Mode (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode)
- AIORET_TYPE CTR_8254Load (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ReadModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue, unsigned short *pReadValue)
- AIORET_TYPE CTR_8254Read (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short *pReadValue)
- AIORET_TYPE CTR_8254ReadAll (unsigned long DeviceIndex, unsigned short *pData)
- AIORET_TYPE CTR_8254ReadStatus (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short *pReadValue, unsigned char *pStatus)
- AIORET_TYPE CTR_StartOutputFreq (unsigned long DeviceIndex, unsigned long BlockIndex, double *pHz)
- AIORET_TYPE CTR_8254SelectGate (unsigned long DeviceIndex, unsigned long GateIndex)
- AIORET_TYPE CTR_8254ReadLatched (unsigned long DeviceIndex, unsigned short *pData)
- AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock *, AIOUSBDevice *deviceDesc, unsigned int)
- AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock *config, AIOUSBDevice *deviceDesc)
- AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock *config)
- void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock *configBlock, AIOUSBDevice *deviceDesc)
- AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock *config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock *config, unsigned reg, unsigned char value)
- AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock *config, unsigned channel)
- AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock *config, unsigned channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock *config, int clock_rate)
- AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock *config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock *config, unsigned char startChannel)
- AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock *config, unsigned char endChannel)
- AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock *config, unsigned startChannel, unsigned endChannel, unsigned gainCode)

  INTERNAL_DOCUMENTATION.

- AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock *config, ADCalMode calMode)
- AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock *config)
- char * ADCConfigBlockToYAML (ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock *config, unsigned overSample)
- AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock *config, unsigned timeout)
- AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock *config)
- AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock *config, unsigned triggerMode)
- AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock *config, int ref)

  Sets the Timer reference.

- AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock *config, AIOUSB_BOOL val)
- AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock *config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock *config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock *to, ADCConfigBlock *from)
- AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock *obj, AIOUSBDevice *dev)
- AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock *obj, AIOUSBDevice *dev)
- AIOUSBDevice * ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock *obj, AIORET_TYPE *res)
- AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock *config, AIOUSBDevice *dev)

  initializes an ADCConfigBlock using parameters from the AIOUSBDevice

- AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock ∗obj, AIOUSB_BOOL testing)
- AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock ∗obj, unsigned size)
- AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock ∗obj, AIOUSB_BOOL debug)
- AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock ∗obj)
- char ∗ ADCConfigBlockToJSON (ADCConfigBlock ∗config)

  INTERNAL_DOCUMENTATION.
- ADCConfigBlock ∗ NewADCConfigBlockFromJSON (const char ∗str)
- AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗config)
- AIOUSB_BOOL is_all_digits (char ∗str)
- AIOChannelMask ∗ NewAIOChannelMask (unsigned size)

  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)

  Destructor for the AIOChannelMask object.
- AIOChannelMask ∗ NewAIOChannelMaskFromStr (const char ∗bitfields)

  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask ∗ NewAIOChannelMaskFromChr (const char bits)
- char ∗ AIOChannelMaskToString (AIOChannelMask ∗mask)

  Returns a string representation for the AIOChannel Bit mask in question.
- char ∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗obj, unsigned index)

  Returns a mask for the index in question.
- char ∗ AIOChannelMaskGetMask (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗mask, char ∗val, unsigned index)

  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗mask)

  Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗mask)

  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)

  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)

  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗mask, unsigned field)

  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗mask, char field, unsigned index)

  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗mask, const char ∗bitfields)

  Rely on the base type to determine the sizes.
- 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)
- AIOBuf ∗ NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf ∗type)
- AIORET_TYPE AIOBufGetSize (AIOBuf ∗buf)
- AIOBufType AIOBufGetType (AIOBuf ∗buf)
- void ∗ AIOBufGetRaw (AIOBuf ∗buf)
- AIORET_TYPE AIOBufRead (AIOBuf ∗buf, void ∗tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf ∗buf, void ∗frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf ∗buf)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator ∗biter)

  Returns a value from the current interator.
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗biter)
- void AIOBufIteratorNext (AIOBufIterator ∗biter)
- AIOCountsConverter ∗ NewAIOCountsConverterWithBuffer (void ∗buf, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverter (unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverterFromAIOContinuousBuf (void ∗buf)
- AIOCountsConverter ∗ NewAIOCountsConverterWithScanLimiter (void ∗buf, unsigned num_scans, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)

- void AIOCountsConverterReset (AIOCountsConverter *cc)
- void DeleteAIOCountsConverter (AIOCountsConverter *ccv)
- AIORET_TYPE AIOCountsConverterConvertNScans (AIOCountsConverter *cc, int num_scans)
- AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (AIOCountsConverter *cc)
- AIORET_TYPE AIOCountsConverterConvert (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIORET_TYPE AIOCountsConverterConvertFifo (AIOCountsConverter *cc, void *tobuf, void *frombuf, unsigned num_bytes)
- AIOGainRange * NewAIOGainRangeFromADCConfigBlock (ADCConfigBlock *adc)
- void DeleteAIOGainRange (AIOGainRange *)
- 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)
- 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.
- 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 ()
- 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)
- 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)
- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double *voltages)

  Preferred way to get immediate scan readings.
- AIORESULT ADC_RangeAll (unsigned long DeviceIndex, unsigned char *pGainCodes, unsigned long bSingleEnded)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double *singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short *pBuf)

  This simple function takes one scan of A/D data, in counts.
- AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[])

  Performs a scan and averages the voltage values.
- AIORESULT ADC_GetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_SetConfig (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize)
- AIORESULT ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- AIORESULT ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)

- AIORESULT ADC_SetCal (unsigned long DeviceIndex, const char ∗CalFileName)
- AIORESULT ADC_QueryCal (unsigned long DeviceIndex)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long ProductID)
- AIORESULT ADC_Initialize (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize, const char ∗CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void ∗pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkPoll (unsigned long DeviceIndex, unsigned long ∗BytesLeft)
- AIORESULT ADC_InitFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- AIORESULT ADC_ResetFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- AIORESULT ADC_GetFastITScanV (unsigned long DeviceIndex, double ∗pData)
- AIORESULT ADC_GetITScanV (unsigned long DeviceIndex, double ∗pBuf)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- AIORESULT WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock ∗config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock ∗config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock ∗config, unsigned channel)
- AIORESULT AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[], int numPoints, unsigned short returnCalTable[], const char ∗saveFileName)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetAllGainCodeAndDiffMode instead")
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetDifferentialMode")
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetCalMode")
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock ∗config, unsigned calMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetCalMode")
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗config, unsigned overSample) ACCES_DEPRECATED("Please use ADCConfigBlockSetOversample")
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetOversample")
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetTriggerMode")
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock ∗config, unsigned triggerMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetTriggerMode")
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock ∗config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- unsigned long AIOUSB_SetStreamingBlockSize (unsigned long DeviceIndex, unsigned long BlockSize)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- AIOBuf ∗ CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[], const char ∗saveFileName)

  Performs automatic calibration of the ADC.
- unsigned char ∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗config)
- void AIOUSB_SetRegister (ADConfigBlock ∗cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock ∗cb, unsigned int Register)
- 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)
- 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)
- 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)
- AIORESULT AIOUSB_GetDeviceByProductID (int minProductID, int maxProductID, int maxDevices, int *deviceList)
- AIORESULT GetDeviceBySerialNumber (uint64_t pSerialNumber)
- AIORESULT GetDeviceSerialNumber (unsigned long DeviceIndex, uint64_t *pSerialNumber)
- AIORESULT AIOUSB_GetDeviceProperties (unsigned long DeviceIndex, DeviceProperties *properties)

  AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()
- 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)
- 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.
- USBDevice * NewUSBDevice (libusb_device *dev, libusb_device_handle *handle)
- void DeleteUSBDevice (USBDevice *dev)
- USBDevice * CopyUSBDevice (USBDevice *usb)
- AIOEither InitializeUSBDevice (USBDevice *usb, LIBUSBArgs *args)

  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device **deviceList, USBDevice **devs, int *size)
- void DeleteUSBDevices (USBDevice *devs)
- int USBDeviceClose (USBDevice *dev)
- int USBDeviceGetIdProduct (USBDevice *device)
- int USBDeviceFetchADCConfigBlock (USBDevice *device, ADCConfigBlock *config)

- int USBDevicePutADCConfigBlock (USBDevice *usb, ADCConfigBlock *configBlock)
- int usb_control_transfer (USBDevice *dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char *data, uint16_t wLength, unsigned int timeout)
- int usb_bulk_transfer (USBDevice *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *actual_length, unsigned int timeout)
- int usb_request (USBDevice *dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t 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)
- 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)

Variables

- struct ad_gain_pairs AD_GAIN_CODE_STRINGS []
- int aio_channel_range_error
- unsigned long AIOUSB_INIT_PATTERN

34.206.1 Macro Definition Documentation

#define MATLAB

#define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain_pairs) - 1))

34.206.2 Typedef Documentation

typedef unsigned long AIORESULT

typedef int64_t AIORET_TYPE

typedef struct aio_channel_range AIOChannelRange

typedef struct aio_counts_converter AIOCountsConverter

34.206.3 Function Documentation

DIOBuf* NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf * or Null if failure

DIOBuf* NewDIOBufFromChar (const char * *ary*, int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf* NewDIOBufFromBinStr (const char * *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

void DeleteDIOBuf (DIOBuf * *buf*)

DIOBuf* DIOBufReplaceString (DIOBuf * *buf*, char * *ary*, int *size_array*)

Replaces the content of the buffer buf with the new array , of size size *.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf* DIOBufReplaceBinString (DIOBuf * *buf*, char * *bitstr*)

char* DIOBufToHex (DIOBuf * *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char* DIOBufToBinary (DIOBuf * *buf*)

char* DIOBufToInvertedBinary (DIOBuf * *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf,* unsigned *size*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

AIORET_TYPE DIOBufSetIndex (DIOBuf ∗ *buf,* int *index,* unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf ∗ *buf,* int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char ∗ *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char *value*)

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask ∗ *mask,* DIOBuf ∗ *buf*)

unsigned long DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void ∗ *pOutMask,* void ∗ *pData*)

unsigned long DIO_ConfigureEx (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pData,* void ∗ *pTristateMask*)

unsigned long DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pTristateMask*)

unsigned long DIO_WriteAll (unsigned long *DeviceIndex,* void ∗ *pData*)

unsigned long DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

unsigned long DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf * *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* void * *buf*)

unsigned long DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char * *buf,* unsigned *size*)

unsigned long DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char * *pdat*)

unsigned long DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char * *bit*)

unsigned long DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

unsigned long DIO_StreamClose (unsigned long *DeviceIndex*)

unsigned long DIO_StreamSetClocks (unsigned long *DeviceIndex,* double * *ReadClockHz,* double * *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```


```
* 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
convert parameter types to those that libusb likes

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

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

build index of result codes

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex,* const char * *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex,* FIFO_Method *Method*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock * *config,* unsigned long *DeviceIndex,* AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

AIORESULT GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_read*)

Performs basic low level USB vendor request.

Returns

AIORESULT GenericVendorWrite (unsigned long *DeviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void * *bufData,* unsigned long * *bytes_write*)

Performs a generic vendor USB write.

AIORESULT AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

AIOContinuousBuf* NewAIOContinuousBuf (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_-oversamples	
base_size	

Returns

AIOContinuousBuf* NewAIOContinuousBufForCounts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels*)

AIOContinuousBuf* NewAIOContinuousBufForVolts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels,* unsigned *num_oversamples*)

AIORET_TYPE DeleteAIOContinuousBuf (**AIOContinuousBuf** * *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufInitConfiguration (**AIOContinuousBuf** * *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value < 0 if not.

AIORET_TYPE AIOContinuousBufInitADCConfigBlock (**AIOContinuousBuf** * *buf,* unsigned *size,* **ADGainCode** *gainCode,* **AIOUSB_BOOL** *diffMode,* unsigned char *os,* **AIOUSB_BOOL** *dfs*)

AIOUSB_WorkFn AIOContinuousBufGetCallback (**AIOContinuousBuf** * *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetCallback (**AIOContinuousBuf** * *buf,* **void** *(*)(**void** *object) *work*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (**AIOContinuousBuf** * *buf,* unsigned *sblksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (**AIOContinuousBuf** * *buf*)

ADCConfigBlock* AIOContinuousBufGetADCConfigBlock (**AIOContinuousBuf** * *buf*)

AIORET_TYPE AIOContinuousBufSetNumberChannels (**AIOContinuousBuf** * *buf,* unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE AIOContinuousBufGetNumberChannels (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *num_oversamples*)

AIORET_TYPE AIOContinuousBufNumberChannels (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetBaseSize (**AIOContinuousBuf** ∗ *buf,* size_t *newbase*)

AIORET_TYPE AIOContinuousBufGetBaseSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (**AIOContinuousBuf** ∗ *buf,* **uint16_t** *new_unit_size*)

AIORET_TYPE AIOContinuousBufGetUnitSize (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBufSetChannelMask (**AIOContinuousBuf** ∗ *buf,* **AIOChannelMask** ∗ *mask*)

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBufNumberSignals (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufResetDevice (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTimeout (**AIOContinuousBuf** ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetNumberScans (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberScans (**AIOContinuousBuf** ∗ *buf,* **int64_t** *num_scans*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf * *buf*)

Todo Fix this to use condition variable

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf * *buf,* void * *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf * *buf,* void * *tobuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf * *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf * *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf * *buf,* AIOCmd * *cmd,* AIORET_TYPE(*)(AIOContinuousBuf *buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd *cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf * *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf * *buf,* unsigned short * *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

- **Parameters**

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf * *buf,* unsigned short * *read_buf,* unsigned *read_buf_size*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf * *buf,* unsigned short * *read_buf,* unsigned *tmpbuffer_size,* int64_t *num_scans*)

will read in an integer number of scan counts if there is room.

- **Parameters**

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufSetCountsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (**AIOContinuousBuf** ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufSetClock (**AIOContinuousBuf** ∗ *buf,* unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufEnd (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gainCode*)

AIORET_TYPE AIOContinuousBufRead (**AIOContinuousBuf** ∗ *buf,* **AIOBufferType** ∗ *readbuf,* unsigned *readbufsize,* unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure ∗.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

If number is positive, it is the number of bytes that have been read.

AIORET_TYPE AIOContinuousBufWrite (**AIOContinuousBuf** ∗ *buf,* **AIOBufferType** ∗ *writebuf,* unsigned *wrbufsize,* unsigned *size,* **AIOContinuousBufMode** *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (**AIOContinuousBuf** ∗ *buf,* unsigned short ∗ *data,* unsigned *datasize,* unsigned *size,* **AIOContinuousBufMode** *flag*)

AIORET_TYPE AIOContinuousBufCleanup (**AIOContinuousBuf** ∗ *buf*)

char∗ AIOContinuousBufToJSON (**AIOContinuousBuf** ∗ *buf*)

AIOContinuousBuf∗ NewAIOContinuousBufFromJSON (**const** char ∗ *json_string*)

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short * *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short * *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short * *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short * *pReadValue,* unsigned char * *pStatus*)

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double * *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short * *pData*)

AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock * *config,* AIOUSBDevice * *deviceDesc,* unsigned *size*)

Parameters

config	
deviceDesc	
size	

AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock * *config,* AIOUSBDevice * *deviceDesc*)

Parameters

config	
deviceDesc	

Returns

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock * *config*)

void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock * *configBlock,* AIOUSBDevice * *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock * *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock * *config,* unsigned *reg,* unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock * *config,* unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock * *config,* unsigned *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetClockRate (**ADCConfigBlock** ∗ *config,* int *clock_rate*)

AIORET_TYPE ADCConfigBlockGetClockRate (**ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockSetScanRange (**ADCConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

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*)

AIOChannelMask* NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask * *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

AIOChannelMask* NewAIOChannelMaskFromStr (const char * *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ NewAIOChannelMaskFromChr (const char *bits*)

char∗ AIOChannelMaskToString (AIOChannelMask ∗ *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

Todo Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

Todo Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗ *obj,* unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE
Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ AIOChannelMaskGetMask (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗ *obj,* char ∗ *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
∗*tmp*	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗ *obj*)

Returns channels that are set to High (not low) ∗.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask * *mask*)

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask * *mask*)

Gives the size of the given BitMask.

AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask * *mask,* int * *pos*)

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask * *mask,* int * *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask * *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask * *mask,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask * *obj,* const char * *bitfields*)

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

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*)

AIOBuf* NewAIOBuf (AIOBufType *type,* size_t *size*)

AIORET_TYPE DeleteAIOBuf (AIOBuf * *type*)

AIORET_TYPE AIOBufGetSize (AIOBuf * *buf*)

AIOBufType AIOBufGetType (AIOBuf * *buf*)

void* AIOBufGetRaw (AIOBuf * *buf*)

AIORET_TYPE AIOBufRead (AIOBuf * *buf,* void * *tobuf,* size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (AIOBuf * *buf,* void * *frombuf,* size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (**AIOBuf** ∗ *buf*)

AIOEither AIOBufIteratorGetValue (**AIOBufIterator** ∗ *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

AIOUSB_BOOL AIOBufIteratorIsValid (**AIOBufIterator** ∗ *biter*)

void AIOBufIteratorNext (**AIOBufIterator** ∗ *biter*)

AIOCountsConverter∗ NewAIOCountsConverterWithBuffer (**void** ∗ *buf,* unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverter (unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverterFromAIOContinuousBuf (**void** ∗ *buf*)

AIOCountsConverter∗ NewAIOCountsConverterWithScanLimiter (**void** ∗ *buf,* unsigned *num_scans,* unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

void AIOCountsConverterReset (**AIOCountsConverter** ∗ *cc*)

void DeleteAIOCountsConverter (**AIOCountsConverter** ∗ *ccv*)

AIORET_TYPE AIOCountsConverterConvertNScans (**AIOCountsConverter** ∗ *cc,* int *num_scans*)

AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (**AIOCountsConverter** ∗ *cc*)

AIORET_TYPE AIOCountsConverterConvert (**AIOCountsConverter** ∗ *cc,* **void** ∗ *tobuf,* **void** ∗ *frombuf,* unsigned *num_bytes*)

AIORET_TYPE AIOCountsConverterConvertFifo (**AIOCountsConverter** ∗ *cc,* **void** ∗ *tobufptr,* **void** ∗ *frombufptr,* unsigned *num_counts*)

Parameters

cc	Counts converter object
tobufptr	ToFifo (double)
frombufptr	From Fifo (unsigned short)
num_counts	number of counts to convert

Returns

Number of tobufptr objects that have been created

AIOGainRange* NewAIOGainRangeFromADCConfigBlock (**ADCConfigBlock** * *adc*)

void DeleteAIOGainRange (**AIOGainRange** *)

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*)

AIODeviceQuery* NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery * object

AIORET_TYPE DeleteAIODeviceQuery (**AIODeviceQuery** * *devq*)

Destructor for AIODeviceQuery *.

Parameters

devq	AIODeviceQuery *

Returns

>= 0, success, otherwise error

char* AIODeviceQueryToStr (**AIODeviceQuery** * *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

char* AIODeviceQueryToRepr (**AIODeviceQuery** * *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery *

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (**AIODeviceQuery** * *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the product ID in question, otherwise error

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery * *devq*)

Returns the strlenght of the Device name of the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0, the name length in question, otherwise error

char* AIODeviceQueryGetName (AIODeviceQuery * *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery *

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery * *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery * *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery *

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery * *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery *

Returns

>= 0 index , otherwise error

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;'

Todo Rely on Global Header files for the functionality of devices / cards as opposed to hard coding

Note

populate device table so users can use diFirst and diOnly immediately; be sure to call PopulateDeviceTable() after 'aiousbInit = AIOUSB_INIT_PATTERN;'

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long * *products,* int *length*)

AIORESULT AIODeviceTableClearDevices (void)

AIORESULT ClearDevices (void)

AIOUSBDevice* AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

AIOUSBDevice* AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

USBDevice* AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT * *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice * A Usb handle that can be used for USB transactions

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex,* unsigned long * *pPID,* unsigned long * *pNameSize,* char * *pName,* unsigned long * *pDIOBytes,* unsigned long * *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if $<$ 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid
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

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

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

AIOUSB_BOOL AIOUSB_IsInit ( )

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

void AIODeviceTableInit (void)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

void CloseAllDevices (void)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIOUSBGetError ( )

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*)

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*)

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double ∗ *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORESULT ADC_RangeAll (unsigned long *DeviceIndex,* unsigned char * *pGainCodes,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double * *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short * *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[ ]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment
The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

AIORESULT ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char * *ConfigBuf,* unsigned long * *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

AIORESULT ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char ∗ *pConfigBuf,* unsigned long ∗ *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

validate settings

AIORESULT ADC_Range1 (unsigned long *DeviceIndex,* unsigned long *ADChannel,* unsigned char *GainCode,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

AIORESULT ADC_ADMode (unsigned long *DeviceIndex,* unsigned char *TriggerMode,* unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

AIORESULT ADC_SetCal (unsigned long *DeviceIndex,* const char * *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIORESULT ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *ProductID*)

AIORESULT ADC_Initialize (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize,* const char * *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

AIORESULT ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIORESULT ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

AIORESULT ADC_InitFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

AIORESULT ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

AIORESULT ADC_GetFastITScanV (unsigned long *DeviceIndex,* double * *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

AIORESULT ADC_GetITScanV (unsigned long *DeviceIndex,* double * *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex,* unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

AIORESULT WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ReadConfigBlock (unsigned long *DeviceIndex,* **AIOUSB_BOOL** *forceRead*)

Parameters

DeviceIndex	
forceRead	

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (**ADConfigBlock** ∗ *config,* unsigned *gainCode,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (**const ADConfigBlock** ∗ *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (**ADConfigBlock** ∗ *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (**const ADConfigBlock** ∗ *config,* unsigned *channel*)

Parameters

config	
channel	

Returns

AIORESULT AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex,* **const** double *points[],* int *numPoints,* unsigned short *returnCalTable[],* **const** char ∗ *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                               ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                               ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock * *config,* unsigned *channel,* AIOUSB_BOOL *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config,* unsigned *calMode*)

Parameters

config	
calMode	

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock * *config,* unsigned *overSample*)

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock * *config*)

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock * *config*, unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const ADConfigBlock * *config*)

unsigned AIOUSB_GetEndChannel (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock * *config*, unsigned *startChannel*, unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex*, unsigned long *BlockSize*)

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex*, AIOUSB_BOOL *discard*)

Parameters

DeviceIndex	
discard	

Returns

AIOBuf* CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

Parameters

DeviceIndex	

Returns

AIOBuf * new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex*, AIOUSB_BOOL *autoCal*, unsigned short *returnCalTable[]*, const char * *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

unsigned char∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗ *config*)

void AIOUSB_SetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register*)

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

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*)

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)

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*)

AIORESULT AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int * *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices * 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

< 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

else, even if we get an error requesting the serial number from this device, keep searching

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t * *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

AIORESULT AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties * *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

AIORET_TYPE AIOUSB_ListDevices ( )

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_FindDevices (int ** *where,* int * *length,* AIOUSB_BOOL(*)(AIOUSBDevice *dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ** *where,* int * *length,* AIOProductGroup * *pg*)

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist * *indices,* AIOProductGroup * *pg*)

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

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
> 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*)

AIOCommandLineOptions* NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

> AIOCommandLineOptions * a new object containing the default settings

AIOCommandLineOptions* NewAIOCommandLineOptionsFromDefaultOptions (**const AIOCommandLineOptions** * *orig*)

AIOCommandLineOptions* AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions* AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (**AIOCommandLineOptions** * *opts*)

AIORET_TYPE AIOProcessCmdline (**AIOCommandLineOptions** * *options,* int *argc,* char ** *argv*)

A simplified command line parsing library for.

Parameters

options	AIOCommandLineOptions object that holds overridden cmd line options
argc	Number of command line arguments
argv	Array of strings to the command line arguments

Returns

AIORET_TYPE AIOProcessCommandLine (**AIOCommandLineOptions** * *options,* int * *argc,* char ** *argv*)

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

AIOChannelRangeTmp* **AIOGetChannelRange** (**char** * *optarg*)

void AIOPrintUsage (**int** *argc,* **char** ** *argv,* **struct option** * *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIORET_TYPE DeleteAIOCommandLineOptions (**AIOCommandLineOptions** * *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

AIORET_TYPE AIOCommandLineOptionsListDevices (**AIOCommandLineOptions** * *options,* **intlist** * *indices*)

Lists any devices that were matched and then lists which index was specified.

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

>= AIOUSB_SUCCESS if devices have been found, < 0 if no devices found

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (**ADCConfigBlock** * *config,* **AIOCommandLineOptions** * *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

>= AIOUSB_SUCCESS is successful, < 0 if there was an error

const AIOCommandLineOptions* **AIO_SCRIPTING_OPTIONS** (**void**)

const AIOCommandLineOptions* **AIO_CMDLINE_OPTIONS** (**void**)

34.206.4 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

unsigned long AIOUSB_INIT_PATTERN

34.207 lib/ndkbuild/jni/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)

34.207.1 Function Documentation

AIOEither InitializeUSBDevice (USBDevice ∗ *usb,* LIBUSBArgs ∗ *args*)

Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

USBDevice∗ NewUSBDevice (libusb_device ∗ *dev,* libusb_device_handle ∗ *handle*)

USBDevice∗ CopyUSBDevice (USBDevice ∗ *usb*)

int USBDeviceClose (USBDevice ∗ *usb*)

AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗ *deviceList,* USBDevice ∗∗ *devs,* int ∗ *size*)

AIORET_TYPE AddDevice (int * *size,* int *index,* libusb_device ** *deviceList,* USBDevice ** *devs,* struct libusb_device_descriptor * *libusbDeviceDesc*)

int USBDeviceGetIdProduct (USBDevice * *device*)

void DeleteUSBDevices (USBDevice * *devices*)

void DeleteUSBDevice (USBDevice * *dev*)

int USBDeviceSetDebug (USBDevice * *usb,* AIOUSB_BOOL *debug*)

libusb_device_handle* USBDeviceGetUSBDeviceHandle (USBDevice * *usb*)

libusb_device_handle* get_usb_device (USBDevice * *dev*)

int USBDeviceFetchADCConfigBlock (USBDevice * *usb,* ADCConfigBlock * *configBlock*)

int USBDevicePutADCConfigBlock (USBDevice * *usb,* ADCConfigBlock * *configBlock*)

int usb_control_transfer (USBDevice * *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char * *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_bulk_transfer (USBDevice * *usb,* unsigned char *endpoint,* unsigned char * *data,* int *length,* int * *actual_length,* unsigned int *timeout*)

This function is intended to improve upon libusb_bulk_transfer() by receiving or transmitting packets until the entire transfer request has been satisfied; it intentionally restarts the timeout each time a packet is received, so the timeout parameter specifies the longest permitted delay between packets, not the total time to complete the transfer request

Note

> even if we get a timeout, some data may have been transferred; if so, then this timeout is not an error; if we get a timeout and no data was transferred, then treat it as an error condition

int usb_request (USBDevice * *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char * *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_reset_device (USBDevice * *usb*)

34.208 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)

34.208.1 Function Documentation

AIOEither InitializeUSBDevice (USBDevice * *usb,* LIBUSBArgs * *args*)

Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

USBDevice* NewUSBDevice (libusb_device * *dev,* libusb_device_handle * *handle*)

USBDevice* CopyUSBDevice (USBDevice * *usb*)

int USBDeviceClose (USBDevice * *usb*)

AIORET_TYPE AddAllACCESUSBDevices (libusb_device ** *deviceList,* USBDevice ** *devs,* int * *size*)

AIORET_TYPE AddDevice (int * *size,* int *index,* libusb_device ** *deviceList,* USBDevice ** *devs,* struct libusb_device_descriptor * *libusbDeviceDesc*)

int USBDeviceGetIdProduct (USBDevice * *device*)

void DeleteUSBDevices (USBDevice * *devices*)

void DeleteUSBDevice (USBDevice * *dev*)

int USBDeviceSetDebug (USBDevice * *usb,* AIOUSB_BOOL *debug*)

libusb_device_handle* USBDeviceGetUSBDeviceHandle (USBDevice * *usb*)

libusb_device_handle* get_usb_device (USBDevice * *dev*)

int USBDeviceFetchADCConfigBlock (USBDevice * *usb,* ADCConfigBlock * *configBlock*)

int USBDevicePutADCConfigBlock (USBDevice * *usb,* ADCConfigBlock * *configBlock*)

int usb_control_transfer (USBDevice * *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char * *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_bulk_transfer (USBDevice * *usb,* unsigned char *endpoint,* unsigned char * *data,* int *length,* int * *actual_length,* unsigned int *timeout*)

This function is intended to improve upon libusb_bulk_transfer() by receiving or transmitting packets until the entire transfer request has been satisfied; it intentionally restarts the timeout each time a packet is received, so the timeout parameter specifies the longest permitted delay between packets, not the total time to complete the transfer request

Note

> even if we get a timeout, some data may have been transferred; if so, then this timeout is not an error; if we get a timeout and no data was transferred, then treat it as an error condition

int usb_request (USBDevice * *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char * *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_reset_device (USBDevice * *usb*)

34.209 lib/ndkbuild/jni/USBDevice.h File Reference

```
#include <stdint.h>
#include <libusb.h>
#include <stdlib.h>
#include "ADCConfigBlock.h"
#include "AIOEither.h"
```

Data Structures

- struct USBDevice
- struct aiousb_libusb_args

Macros

- #define INTERNAL_METHOD(NAME, RETVAL,...) RETVAL (∗NAME)(__VA_ARGS__)

Typedefs

- typedef struct USBDevice USBDevice
- typedef struct aiousb_libusb_args LIBUSBArgs

Functions

- USBDevice ∗ NewUSBDevice (libusb_device ∗dev, libusb_device_handle ∗handle)
- void DeleteUSBDevice (USBDevice ∗dev)
- USBDevice ∗ CopyUSBDevice (USBDevice ∗usb)
- AIOEither InitializeUSBDevice (USBDevice ∗usb, LIBUSBArgs ∗args)

  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)
- void DeleteUSBDevices (USBDevice ∗devs)
- int USBDeviceClose (USBDevice ∗dev)
- int USBDeviceGetIdProduct (USBDevice ∗device)
- int USBDeviceFetchADCConfigBlock (USBDevice ∗device, ADCConfigBlock ∗config)
- int USBDevicePutADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_bulk_transfer (USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int usb_request (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t 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)

34.209.1 Macro Definition Documentation

#define INTERNAL_METHOD(*NAME, RETVAL, ...*) RETVAL (∗NAME)(__VA_ARGS__)

34.209.2 Typedef Documentation

typedef struct USBDevice USBDevice

typedef struct aiousb_libusb_args LIBUSBArgs

34.209.3 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
> 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*)

34.210 lib/USBDevice.h File Reference

```
#include <stdint.h>
#include <libusb.h>
#include <stdlib.h>
#include "ADCConfigBlock.h"
#include "AIOEither.h"
```

Data Structures

- struct USBDevice
- struct aiousb_libusb_args

Macros

- #define INTERNAL_METHOD(NAME, RETVAL,...) RETVAL (∗NAME)(__VA_ARGS__)

Typedefs

- typedef struct USBDevice USBDevice
- typedef struct aiousb_libusb_args LIBUSBArgs

Functions

- USBDevice ∗ NewUSBDevice (libusb_device ∗dev, libusb_device_handle ∗handle)
- void DeleteUSBDevice (USBDevice ∗dev)
- USBDevice ∗ CopyUSBDevice (USBDevice ∗usb)

- AIOEither InitializeUSBDevice (USBDevice ∗usb, LIBUSBArgs ∗args)
  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)
- void DeleteUSBDevices (USBDevice ∗devs)
- int USBDeviceClose (USBDevice ∗dev)
- int USBDeviceGetIdProduct (USBDevice ∗device)
- int USBDeviceFetchADCConfigBlock (USBDevice ∗device, ADCConfigBlock ∗config)
- int USBDevicePutADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_bulk_transfer (USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int usb_request (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t 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)

34.210.1 Macro Definition Documentation

#define INTERNAL_METHOD(*NAME, RETVAL, ...*) RETVAL (∗NAME)(__VA_ARGS__)

34.210.2 Typedef Documentation

typedef struct USBDevice USBDevice

typedef struct aiousb_libusb_args LIBUSBArgs

34.210.3 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
> 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*)

34.211 lib/ndkbuild/libusb/android/config.h File Reference

Macros

- #define ENABLE_LOGGING
- #define HAVE_DLFCN_H 1
- #define HAVE_GETTIMEOFDAY 1
- #define HAVE_INTTYPES_H 1
- #define OS_LINUX 1
- #define USE_SYSTEM_LOGGING_FACILITY 1
- #define POLL_NFDS_TYPE nfds_t
- #define THREADS_POSIX 1
- #define DEFAULT_VISIBILITY __attribute__((visibility("default")))
- #define HAVE_MEMORY_H 1
- #define HAVE_POLL_H 1
- #define HAVE_SIGNAL_H 1
- #define HAVE_SYS_STAT_H 1
- #define HAVE_SYS_TIME_H 1
- #define HAVE_SYS_TYPES_H 1
- #define HAVE_UNISTD_H 1
- #define HAVE_LINUX_FILTER_H 1
- #define HAVE_LINUX_NETLINK_H 1
- #define HAVE_ASM_TYPES_H 1
- #define HAVE_SYS_SOCKET_H 1

34.211.1 Macro Definition Documentation

#define ENABLE_LOGGING

#define HAVE_DLFCN_H 1

#define HAVE_GETTIMEOFDAY 1

#define HAVE_INTTYPES_H 1

#define OS_LINUX 1

#define USE_SYSTEM_LOGGING_FACILITY 1

#define POLL_NFDS_TYPE nfds_t

#define THREADS_POSIX 1

#define DEFAULT_VISIBILITY __attribute__((visibility("default")))

#define HAVE_MEMORY_H 1

#define HAVE_POLL_H 1

#define HAVE_SIGNAL_H 1

#define HAVE_SYS_STAT_H 1

#define HAVE_SYS_TIME_H 1

#define HAVE_SYS_TYPES_H 1

#define HAVE_UNISTD_H 1

#define HAVE_LINUX_FILTER_H 1

#define HAVE_LINUX_NETLINK_H 1

#define HAVE_ASM_TYPES_H 1

#define HAVE_SYS_SOCKET_H 1

34.212 lib/ndkbuild/libusb/msvc/config.h File Reference

Macros

- #define DEFAULT_VISIBILITY
- #define ENABLE_LOGGING 1
- #define POLL_NFDS_TYPE unsigned int
- #define OS_WINDOWS 1
- #define HAVE_SIGNAL_H 1
- #define HAVE_SYS_TYPES_H 1

34.212.1 Macro Definition Documentation

#define DEFAULT_VISIBILITY

#define ENABLE_LOGGING 1

#define POLL_NFDS_TYPE unsigned int

#define OS_WINDOWS 1

#define HAVE_SIGNAL_H 1

#define HAVE_SYS_TYPES_H 1

34.213 lib/ndkbuild/libusb/Xcode/config.h File Reference

Macros

- #define DEFAULT_VISIBILITY
- #define ENABLE_LOGGING 1
- #define HAVE_GETTIMEOFDAY 1
- #define HAVE_POLL_H 1
- #define HAVE_SYS_TIME_H 1
- #define OS_DARWIN 1
- #define POLL_NFDS_TYPE nfds_t
- #define THREADS_POSIX 1

34.213.1 Macro Definition Documentation

#define DEFAULT_VISIBILITY

#define ENABLE_LOGGING 1

#define HAVE_GETTIMEOFDAY 1

#define HAVE_POLL_H 1

#define HAVE_SYS_TIME_H 1

#define OS_DARWIN 1

#define POLL_NFDS_TYPE nfds_t

#define THREADS_POSIX 1

34.214 lib/ndkbuild/libusb/examples/dpfp.c File Reference

```
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "libusb.h"
```

Macros

- #define EP_INTR (1 | LIBUSB_ENDPOINT_IN)
- #define EP_DATA (2 | LIBUSB_ENDPOINT_IN)
- #define CTRL_IN (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_IN)
- #define CTRL_OUT (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_OUT)
- #define USB_RQ 0x04
- #define INTR_LENGTH 64

Enumerations

- enum {
  MODE_INIT = 0x00, MODE_AWAIT_FINGER_ON = 0x10, MODE_AWAIT_FINGER_OFF = 0x12, MODE_CAPTURE = 0x20,
  MODE_SHUT_UP = 0x30, MODE_READY = 0x80 }
- enum {
  STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_ON = 1, STATE_AWAIT_IRQ_FINGER_DETECTED, STATE_AWAIT_MODE_CHANGE_CAPTURE, STATE_AWAIT_IMAGE,
  STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_OFF, STATE_AWAIT_IRQ_FINGER_REMOVED }

Functions

- int main (void)

34.214.1 Macro Definition Documentation

#define EP_INTR (1 | LIBUSB_ENDPOINT_IN)

#define EP_DATA (2 | LIBUSB_ENDPOINT_IN)

#define CTRL_IN (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_IN)

#define CTRL_OUT (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_OUT)

#define USB_RQ 0x04

#define INTR_LENGTH 64

34.214.2 Enumeration Type Documentation

anonymous enum

Enumerator

MODE_INIT

MODE_AWAIT_FINGER_ON

MODE_AWAIT_FINGER_OFF

MODE_CAPTURE

MODE_SHUT_UP

MODE_READY

anonymous enum

Enumerator

STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_ON

STATE_AWAIT_IRQ_FINGER_DETECTED

STATE_AWAIT_MODE_CHANGE_CAPTURE

STATE_AWAIT_IMAGE

STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_OFF

STATE_AWAIT_IRQ_FINGER_REMOVED

34.214.3 Function Documentation

int main (void)

34.215 lib/ndkbuild/libusb/examples/dpfp_threaded.c File Reference

```
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include "libusb.h"
```

Macros

- #define EP_INTR (1 | LIBUSB_ENDPOINT_IN)
- #define EP_DATA (2 | LIBUSB_ENDPOINT_IN)
- #define CTRL_IN (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_IN)
- #define CTRL_OUT (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_OUT)
- #define USB_RQ 0x04
- #define INTR_LENGTH 64
- #define SEM_NAME "/org.libusb.example.dpfp_threaded"

Enumerations

- enum {
  MODE_INIT = 0x00, MODE_AWAIT_FINGER_ON = 0x10, MODE_AWAIT_FINGER_OFF = 0x12, MODE_CAPTURE = 0x20,
  MODE_SHUT_UP = 0x30, MODE_READY = 0x80 }
- enum {
  STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_ON = 1, STATE_AWAIT_IRQ_FINGER_DETECTED, STATE_AWAIT_MODE_CHANGE_CAPTURE, STATE_AWAIT_IMAGE,
  STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_OFF, STATE_AWAIT_IRQ_FINGER_REMOVED }

Functions

- int main (void)

34.215.1 Macro Definition Documentation

#define EP_INTR (1 | LIBUSB_ENDPOINT_IN)

#define EP_DATA (2 | LIBUSB_ENDPOINT_IN)

#define CTRL_IN (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_IN)

#define CTRL_OUT (LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_ENDPOINT_OUT)

#define USB_RQ 0x04

#define INTR_LENGTH 64

#define SEM_NAME "/org.libusb.example.dpfp_threaded"

34.215.2 Enumeration Type Documentation

anonymous enum

Enumerator

MODE_INIT

MODE_AWAIT_FINGER_ON

MODE_AWAIT_FINGER_OFF

MODE_CAPTURE

MODE_SHUT_UP

MODE_READY

anonymous enum

Enumerator

STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_ON

STATE_AWAIT_IRQ_FINGER_DETECTED

STATE_AWAIT_MODE_CHANGE_CAPTURE

STATE_AWAIT_IMAGE

STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_OFF

STATE_AWAIT_IRQ_FINGER_REMOVED

34.215.3 Function Documentation

int main (**void**)

34.216 lib/ndkbuild/libusb/examples/ezusb.c File Reference

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "libusb.h"
#include "ezusb.h"
```

Data Structures

- struct ram_poke_context

Macros

- #define RW_INTERNAL 0xA0 /∗ hardware implements this one ∗/
- #define RW_MEMORY 0xA3
- #define RETRY_LIMIT 5

Enumerations

- enum ram_mode { , internal_only, skip_internal, skip_external }

Functions

- void logerror (const char ∗format,...) __attribute__((format(printf
- int ezusb_load_ram (libusb_device_handle ∗device, const char ∗path, int fx_type, int img_type, int stage)

Variables

- void int verbose = 1

34.216.1 Macro Definition Documentation

#define RW_INTERNAL 0xA0 /∗ hardware implements this one ∗/

#define RW_MEMORY 0xA3

#define RETRY_LIMIT 5

34.216.2 Enumeration Type Documentation

enum ram_mode

Enumerator

internal_only

skip_internal

skip_external

34.216.3 Function Documentation

void void logerror (const char ∗ *format,* ...)

int ezusb_load_ram (libusb_device_handle ∗ *device,* const char ∗ *path,* int *fx_type,* int *img_type,* int *stage*)

34.216.4 Variable Documentation

void int verbose = 1

34.217 lib/ndkbuild/libusb/examples/ezusb.h File Reference

```
#include <stdbool.h>
```

Data Structures

- struct fx_known_device

Macros

- #define FX_TYPE_UNDEFINED -1
- #define FX_TYPE_AN21 0 /∗ Original AnchorChips parts ∗/
- #define FX_TYPE_FX1 1 /∗ Updated Cypress versions ∗/
- #define FX_TYPE_FX2 2 /∗ USB 2.0 versions ∗/
- #define FX_TYPE_FX2LP 3 /∗ Updated FX2 ∗/
- #define FX_TYPE_FX3 4 /∗ USB 3.0 versions ∗/
- #define FX_TYPE_MAX 5
- #define FX_TYPE_NAMES { "an21", "fx", "fx2", "fx2lp", "fx3" }
- #define IMG_TYPE_UNDEFINED -1
- #define IMG_TYPE_HEX 0 /∗ Intel HEX ∗/
- #define IMG_TYPE_IIC 1 /∗ Cypress 8051 IIC ∗/
- #define IMG_TYPE_BIX 2 /∗ Cypress 8051 BIX ∗/
- #define IMG_TYPE_IMG 3 /∗ Cypress IMG format ∗/
- #define IMG_TYPE_MAX 4
- #define IMG_TYPE_NAMES { "Intel HEX", "Cypress 8051 IIC", "Cypress 8051 BIX", "Cypress IMG format" }
- #define FX_KNOWN_DEVICES

Functions

- int ezusb_load_ram (libusb_device_handle ∗device, const char ∗path, int fx_type, int img_type, int stage)
- int ezusb_load_eeprom (libusb_device_handle ∗device, const char ∗path, int fx_type, int img_type, int config)

Variables

- int verbose

34.217.1 Macro Definition Documentation

#define FX_TYPE_UNDEFINED -1

#define FX_TYPE_AN21 0 /∗ Original AnchorChips parts ∗/

#define FX_TYPE_FX1 1 /∗ Updated Cypress versions ∗/

#define FX_TYPE_FX2 2 /∗ USB 2.0 versions ∗/

#define FX_TYPE_FX2LP 3 /∗ Updated FX2 ∗/

#define FX_TYPE_FX3 4 /∗ USB 3.0 versions ∗/

#define FX_TYPE_MAX 5

#define FX_TYPE_NAMES { "an21", "fx", "fx2", "fx2lp", "fx3" }

#define IMG_TYPE_UNDEFINED -1

#define IMG_TYPE_HEX 0 /∗ Intel HEX ∗/

#define IMG_TYPE_IIC 1 /∗ Cypress 8051 IIC ∗/

#define IMG_TYPE_BIX 2 /∗ Cypress 8051 BIX ∗/

#define IMG_TYPE_IMG 3 /∗ Cypress IMG format ∗/

#define IMG_TYPE_MAX 4

#define IMG_TYPE_NAMES { "Intel HEX", "Cypress 8051 IIC", "Cypress 8051 BIX", "Cypress IMG format" }

#define FX_KNOWN_DEVICES

Value:

```
{ \
        { 0x0547, 0x2122, FX_TYPE_AN21, "Cypress EZ-USB (2122S)" },\
        { 0x0547, 0x2125, FX_TYPE_AN21, "Cypress EZ-USB (2121S/2125S)" },\
        { 0x0547, 0x2126, FX_TYPE_AN21, "Cypress EZ-USB (2126S)" },\
        { 0x0547, 0x2131, FX_TYPE_AN21, "Cypress EZ-USB (2131Q/2131S/2135S)" },\
        { 0x0547, 0x2136, FX_TYPE_AN21, "Cypress EZ-USB (2136S)" },\
        { 0x0547, 0x2225, FX_TYPE_AN21, "Cypress EZ-USB (2225)" },\
        { 0x0547, 0x2226, FX_TYPE_AN21, "Cypress EZ-USB (2226)" },\
        { 0x0547, 0x2235, FX_TYPE_AN21, "Cypress EZ-USB (2235)" },\
        { 0x0547, 0x2236, FX_TYPE_AN21, "Cypress EZ-USB (2236)" },\
        { 0x04b4, 0x6473, FX_TYPE_FX1, "Cypress EZ-USB FX1" },\
        { 0x04b4, 0x8613, FX_TYPE_FX2LP, "Cypress EZ-USB FX2LP (68013A/68014A/68015A/68016A)"
      }, \
        { 0x04b4, 0x00f3, FX_TYPE_FX3, "Cypress FX3" },\
}
```

34.217.2 Function Documentation

int ezusb_load_ram (**libusb_device_handle** ∗ *device,* **const** char ∗ *path,* int *fx_type,* int *img_type,* int *stage*)

int ezusb_load_eeprom (**libusb_device_handle** ∗ *device,* **const** char ∗ *path,* int *fx_type,* int *img_type,* int *config*)

34.217.3 Variable Documentation

int verbose

34.218 lib/ndkbuild/libusb/examples/fxload.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdarg.h>
#include <sys/types.h>
#include <getopt.h>
#include "libusb.h"
#include "ezusb.h"
#include <syslog.h>
#include <strings.h>
```

Macros

- #define FXLOAD_VERSION (__DATE__ " (libusb)")
- #define ARRAYSIZE(A) (sizeof(A)/sizeof((A)[0]))
- #define FIRMWARE 0
- #define LOADER 1

Functions

- void logerror (const char *format,...) __attribute__((format(__printf__
- int main (int argc, char *argv[])

34.218.1 Macro Definition Documentation

#define FXLOAD_VERSION (__DATE__ " (libusb)")

#define ARRAYSIZE(*A*) (sizeof(A)/sizeof((A)[0]))

#define FIRMWARE 0

#define LOADER 1

34.218.2 Function Documentation

void logerror (const char * *format, ...*)

int main (int *argc,* char * *argv[]*)

34.219 lib/ndkbuild/libusb/examples/getopt/getopt.c File Reference

```
#include <stdio.h>
#include "getopt.h"
#include <strings.h>
```

Macros

- #define const
- #define GETOPT_INTERFACE_VERSION 2
- #define SWAP_FLAGS(ch1, ch2)
- #define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

Enumerations

- enum { REQUIRE_ORDER, PERMUTE, RETURN_IN_ORDER }

Functions

- char * getenv ()
- int getopt (int argc, char *const *argv, const char *optstring)

Variables

- char * optarg
- int optind = 1
- int opterr = 1
- int optopt = '?'

34.219.1 Macro Definition Documentation

#define const

#define GETOPT_INTERFACE_VERSION 2

#define SWAP_FLAGS(*ch1, ch2*)

#define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

34.219.2 Enumeration Type Documentation

anonymous enum

Enumerator

REQUIRE_ORDER

PERMUTE

RETURN_IN_ORDER

34.219.3 Function Documentation

char* getenv ( )

int getopt (int *argc*, char *const * *argv*, const char * *optstring*)

34.219.4 Variable Documentation

char* optarg

int optind = 1

int opterr = 1

int optopt = '?'

34.220 lib/ndkbuild/libusb/examples/getopt/getopt.h File Reference

```
#include <ctype.h>
```

Data Structures

- struct option

Macros

- #define no_argument 0
- #define required_argument 1
- #define optional_argument 2

Functions

- int getopt ()
- int getopt_long ()
- int getopt_long_only ()

Variables

- char * optarg
- int optind
- int opterr
- int optopt

34.220.1 Macro Definition Documentation

#define no_argument 0

#define required_argument 1

#define optional_argument 2

34.220.2 Function Documentation

int getopt ( )

int getopt_long ( )

int getopt_long_only ( )

34.220.3 Variable Documentation

char* optarg

int optind

int opterr

int optopt

34.221 lib/ndkbuild/libusb/examples/getopt/getopt1.c File Reference

```
#include "getopt.h"
#include <stdio.h>
```

Macros

- #define const
- #define GETOPT_INTERFACE_VERSION 2
- #define NULL 0

Functions

- int getopt_long (int argc, char *const *argv, const char *options, const struct option *long_options, int *opt_index)
- int getopt_long_only (int argc, char *const *argv, const char *options, const struct option *long_options, int *opt_index)

34.221.1 Macro Definition Documentation

#define const

#define GETOPT_INTERFACE_VERSION 2

#define NULL 0

34.221.2 Function Documentation

int getopt_long (int *argc*, char *const * *argv*, const char * *options*, const struct option * *long_options*, int * *opt_index*)

int getopt_long_only (int *argc*, char *const * *argv*, const char * *options*, const struct option * *long_options*, int * *opt_index*)

34.222 lib/ndkbuild/libusb/examples/hotplugtest.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "libusb.h"
```

Functions

- int main (int argc, char *argv[])

Variables

- int done = 0
- libusb_device_handle * handle = NULL

34.222.1 Function Documentation

int main (int *argc*, char * *argv[]*)

34.222.2 Variable Documentation

int done = 0

libusb_device_handle* handle = NULL

34.223 lib/ndkbuild/libusb/examples/listdevs.c File Reference

```
#include <stdio.h>
#include "libusb.h"
```

Functions

- int main (void)

34.223.1 Function Documentation

int main (void)

34.224 lib/ndkbuild/libusb/examples/sam3u_benchmark.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <libusb.h>
```

Macros

- #define EP_DATA_IN 0x82
- #define EP_ISO_IN 0x86

Functions

- int main (int argc, char **argv)

34.224.1 Macro Definition Documentation

#define EP_DATA_IN 0x82

#define EP_ISO_IN 0x86

34.224.2 Function Documentation

int main (int *argc*, char ** *argv*)

34.225 lib/ndkbuild/libusb/examples/testlibusb.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "libusb.h"
```

Functions

- int main (int argc, char *argv[])

Variables

- int verbose = 0

34.225.1 Function Documentation

int main (int *argc*, char * *argv[]*)

34.225.2 Variable Documentation

int verbose = 0

34.226 lib/ndkbuild/libusb/examples/xusb.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "libusb.h"
#include <time.h>
```

Data Structures

- struct command_block_wrapper
- struct command_status_wrapper

Macros

- #define msleep(msecs) nanosleep(&(struct timespec){msecs / 1000, (msecs * 1000000) % 1000000000UL}, NULL);
- #define bool int
- #define true (1 == 1)
- #define false (!true)
- #define usb_interface interface
- #define ERR_EXIT(errcode) do { perr(" %s\n", libusb_strerror((enum libusb_error)errcode)); return -1; } while (0)
- #define CALL_CHECK(fcall) do { r=fcall; if (r < 0) ERR_EXIT(r); } while (0);
- #define B(x) (((x)!=0)?1:0)
- #define be_to_int32(buf) (((buf)[0]<<24)|((buf)[1]<<16)|((buf)[2]<<8)|(buf)[3])
- #define RETRY_MAX 5
- #define REQUEST_SENSE_LENGTH 0x12
- #define INQUIRY_LENGTH 0x24

- #define READ_CAPACITY_LENGTH 0x08
- #define HID_GET_REPORT 0x01
- #define HID_GET_IDLE 0x02
- #define HID_GET_PROTOCOL 0x03
- #define HID_SET_REPORT 0x09
- #define HID_SET_IDLE 0x0A
- #define HID_SET_PROTOCOL 0x0B
- #define HID_REPORT_TYPE_INPUT 0x01
- #define HID_REPORT_TYPE_OUTPUT 0x02
- #define HID_REPORT_TYPE_FEATURE 0x03
- #define BOMS_RESET 0xFF
- #define BOMS_GET_MAX_LUN 0xFE
- #define MAX_OS_FD_LENGTH 256

Enumerations

- enum test_type {
  USE_GENERIC, USE_PS3, USE_XBOX, USE_SCSI,
  USE_HID }

Functions

- int main (int argc, char **argv)

34.226.1 Macro Definition Documentation

#define msleep(*msecs*) nanosleep(&(struct timespec){msecs / 1000, (msecs * 1000000) % 1000000000UL}, NULL);

#define bool int

#define true (1 == 1)

#define false (!true)

#define usb_interface interface

#define ERR_EXIT(*errcode*) do { perr(" %s\n", libusb_strerror((enum libusb_error)errcode)); return -1; } while (0)

#define CALL_CHECK(*fcall*) do { r=fcall; if (r < 0) ERR_EXIT(r); } while (0);

#define B(*x*) (((x)!=0)?1:0)

#define be_to_int32(*buf*) (((buf)[0]<<24)|((buf)[1]<<16)|((buf)[2]<<8)|(buf)[3])

#define RETRY_MAX 5

#define REQUEST_SENSE_LENGTH 0x12

#define INQUIRY_LENGTH 0x24

#define READ_CAPACITY_LENGTH 0x08

#define HID_GET_REPORT 0x01

#define HID_GET_IDLE 0x02

#define HID_GET_PROTOCOL 0x03

#define HID_SET_REPORT 0x09

#define HID_SET_IDLE 0x0A

#define HID_SET_PROTOCOL 0x0B

#define HID_REPORT_TYPE_INPUT 0x01

#define HID_REPORT_TYPE_OUTPUT 0x02

#define HID_REPORT_TYPE_FEATURE 0x03

#define BOMS_RESET 0xFF

#define BOMS_GET_MAX_LUN 0xFE

#define MAX_OS_FD_LENGTH 256

34.226.2 Enumeration Type Documentation

enum test_type

Enumerator

USE_GENERIC

USE_PS3

USE_XBOX

USE_SCSI

USE_HID

34.226.3 Function Documentation

int main (int *argc*, char ** *argv*)

34.227 lib/ndkbuild/libusb/libusb/core.c File Reference

```
#include "config.h"
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "libusbi.h"
#include "hotplug.h"
```

Macros

- #define DISCOVERED_DEVICES_SIZE_STEP 8

Functions

- struct discovered_devs * discovered_devs_append (struct discovered_devs *discdevs, struct libusb_device *dev)
- struct libusb_device * usbi_alloc_device (struct libusb_context *ctx, unsigned long session_id)
- void usbi_connect_device (struct libusb_device *dev)
- void usbi_disconnect_device (struct libusb_device *dev)
- int usbi_sanitize_device (struct libusb_device *dev)
- struct libusb_device * usbi_get_device_by_session_id (struct libusb_context *ctx, unsigned long session_id)
- ssize_t API_EXPORTED libusb_get_device_list (libusb_context *ctx, libusb_device ***list)

  Returns a list of USB devices currently attached to the system.
- void API_EXPORTED libusb_free_device_list (libusb_device **list, int unref_devices)

  Frees a list of devices previously discovered using libusb_get_device_list().
- uint8_t API_EXPORTED libusb_get_bus_number (libusb_device *dev)

  Get the number of the bus that a device is connected to.
- uint8_t API_EXPORTED libusb_get_port_number (libusb_device *dev)

  Get the number of the port that a device is connected to.
- int API_EXPORTED libusb_get_port_numbers (libusb_device *dev, uint8_t *port_numbers, int port_numbers_-len)

  Get the list of all port numbers from root for the specified device.
- int API_EXPORTED libusb_get_port_path (libusb_context *ctx, libusb_device *dev, uint8_t *port_numbers, uint8_t port_numbers_len)

  Deprecated please use libusb_get_port_numbers instead.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_get_parent (libusb_device *dev)

  Get the the parent from the specified device.

- uint8_t API_EXPORTED libusb_get_device_address (libusb_device *dev)

  Get the address of the device on the bus it is connected to.
- int API_EXPORTED libusb_get_device_speed (libusb_device *dev)

  Get the negotiated connection speed for a device.
- int API_EXPORTED libusb_get_max_packet_size (libusb_device *dev, unsigned char endpoint)

  Convenience function to retrieve the wMaxPacketSize value for a particular endpoint in the active device configuration.
- int API_EXPORTED libusb_get_max_iso_packet_size (libusb_device *dev, unsigned char endpoint)

  Calculate the maximum packet size which a specific endpoint is capable is sending or receiving in the duration of 1 microframe.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_ref_device (libusb_device *dev)

  Increment the reference count of a device.
- void API_EXPORTED libusb_unref_device (libusb_device *dev)

  Decrement the reference count of a device.
- int usbi_signal_event (struct libusb_context *ctx)
- int usbi_clear_event (struct libusb_context *ctx)
- int API_EXPORTED libusb_open (libusb_device *dev, libusb_device_handle **dev_handle)

  Open a device and obtain a device handle.
- DEFAULT_VISIBILITY libusb_device_handle *LIBUSB_CALL libusb_open_device_with_vid_pid (libusb_context *ctx, uint16_t vendor_id, uint16_t product_id)

  Convenience function for finding a device with a particular `idVendor`/`idProduct` combination.
- void API_EXPORTED libusb_close (libusb_device_handle *dev_handle)

  Close a device handle.
- DEFAULT_VISIBILITY libusb_device *LIBUSB_CALL libusb_get_device (libusb_device_handle *dev_handle)

  Get the underlying device for a device handle.
- int API_EXPORTED libusb_get_configuration (libusb_device_handle *dev_handle, int *config)

  Determine the bConfigurationValue of the currently active configuration.
- int API_EXPORTED libusb_set_configuration (libusb_device_handle *dev_handle, int configuration)

  Set the active configuration for a device.
- int API_EXPORTED libusb_claim_interface (libusb_device_handle *dev_handle, int interface_number)

  Claim an interface on a given device handle.
- int API_EXPORTED libusb_release_interface (libusb_device_handle *dev_handle, int interface_number)

  Release an interface previously claimed with libusb_claim_interface().
- int API_EXPORTED libusb_set_interface_alt_setting (libusb_device_handle *dev_handle, int interface_number, int alternate_setting)

  Activate an alternate setting for an interface.
- int API_EXPORTED libusb_clear_halt (libusb_device_handle *dev_handle, unsigned char endpoint)

  Clear the halt/stall condition for an endpoint.
- int API_EXPORTED libusb_reset_device (libusb_device_handle *dev_handle)

  Perform a USB port reset to reinitialize a device.
- int API_EXPORTED libusb_alloc_streams (libusb_device_handle *dev_handle, uint32_t num_streams, unsigned char *endpoints, int num_endpoints)

  Allocate up to num_streams usb bulk streams on the specified endpoints.
- int API_EXPORTED libusb_free_streams (libusb_device_handle *dev_handle, unsigned char *endpoints, int num_endpoints)

  Free usb bulk streams allocated with libusb_alloc_streams().
- DEFAULT_VISIBILITY unsigned char *LIBUSB_CALL libusb_dev_mem_alloc (libusb_device_handle *dev_handle, size_t length)

  Attempts to allocate a block of persistent DMA memory suitable for transfers against the given device.
- int API_EXPORTED libusb_dev_mem_free (libusb_device_handle *dev_handle, unsigned char *buffer, size_t length)

  Free device memory allocated with libusb_dev_mem_alloc().
- int API_EXPORTED libusb_kernel_driver_active (libusb_device_handle *dev_handle, int interface_number)

  Determine if a kernel driver is active on an interface.
- int API_EXPORTED libusb_detach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Detach a kernel driver from an interface.
- int API_EXPORTED libusb_attach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Re-attach an interface's kernel driver, which was previously detached using libusb_detach_kernel_driver().
- int API_EXPORTED libusb_set_auto_detach_kernel_driver (libusb_device_handle *dev_handle, int enable)

  Enable/disable libusb's automatic kernel driver detachment.
- void API_EXPORTED libusb_set_debug (libusb_context *ctx, int level)

  Set log message verbosity.

- int API_EXPORTED libusb_init (libusb_context **context)

  Initialize libusb.
- void API_EXPORTED libusb_exit (struct libusb_context *ctx)

  Deinitialize libusb.
- int API_EXPORTED libusb_has_capability (uint32_t capability)

  Check at runtime if the loaded library has a given capability.
- void usbi_log_v (struct libusb_context *ctx, enum libusb_log_level level, const char *function, const char *format, va_list args)
- void usbi_log (struct libusb_context *ctx, enum libusb_log_level level, const char *function, const char *format,...)
- DEFAULT_VISIBILITY const char *LIBUSB_CALL libusb_error_name (int error_code)

  Returns a constant NULL-terminated string with the ASCII name of a libusb error or transfer status code.
- DEFAULT_VISIBILITY const struct libusb_version *LIBUSB_CALL libusb_get_version (void)

  Returns a pointer to const struct libusb_version with the version (major, minor, micro, nano and rc) of the running library.

Variables

- struct libusb_context * usbi_default_context = NULL
- usbi_mutex_static_t active_contexts_lock = USBI_MUTEX_INITIALIZER
- struct list_head active_contexts_list

34.227.1 Macro Definition Documentation

#define DISCOVERED_DEVICES_SIZE_STEP 8

34.227.2 Function Documentation

struct discovered_devs* discovered_devs_append (struct discovered_devs * *discdevs*, struct libusb_device * *dev*)

struct libusb_device* usbi_alloc_device (struct libusb_context * *ctx*, unsigned long *session_id*)

void usbi_connect_device (struct libusb_device * *dev*)

void usbi_disconnect_device (struct libusb_device * *dev*)

int usbi_sanitize_device (struct libusb_device * *dev*)

struct libusb_device* usbi_get_device_by_session_id (struct libusb_context * *ctx*, unsigned long *session_id*)

int usbi_signal_event (struct libusb_context * *ctx*)

int usbi_clear_event (struct libusb_context * *ctx*)

void usbi_log_v (struct libusb_context * *ctx*, enum libusb_log_level *level*, const char * *function*, const char * *format*, va_list *args*)

void usbi_log (struct libusb_context * *ctx*, enum libusb_log_level *level*, const char * *function*, const char * *format*, ...)

34.227.3 Variable Documentation

struct libusb_context* usbi_default_context = NULL

usbi_mutex_static_t active_contexts_lock = USBI_MUTEX_INITIALIZER

struct list_head active_contexts_list

34.228 lib/ndkbuild/libusb/libusb/descriptor.c File Reference

```
#include <config.h>
#include <errno.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "libusbi.h"
```

Macros

- #define DESC_HEADER_LENGTH 2
- #define DEVICE_DESC_LENGTH 18
- #define CONFIG_DESC_LENGTH 9
- #define INTERFACE_DESC_LENGTH 9
- #define ENDPOINT_DESC_LENGTH 7
- #define ENDPOINT_AUDIO_DESC_LENGTH 9

Functions

- int usbi_parse_descriptor (const unsigned char ∗source, const char ∗descriptor, void ∗dest, int host_endian)
- int usbi_device_cache_descriptor (libusb_device ∗dev)
- int API_EXPORTED libusb_get_device_descriptor (libusb_device ∗dev, struct libusb_device_descriptor ∗desc)

  Get the USB device descriptor for a given device.
- int API_EXPORTED libusb_get_active_config_descriptor (libusb_device ∗dev, struct libusb_config_descriptor ∗∗config)

  Get the USB configuration descriptor for the currently active configuration.
- int API_EXPORTED libusb_get_config_descriptor (libusb_device ∗dev, uint8_t config_index, struct libusb_config_descriptor ∗∗config)

  Get a USB configuration descriptor based on its index.
- int usbi_get_config_index_by_value (struct libusb_device ∗dev, uint8_t bConfigurationValue, int ∗idx)
- int API_EXPORTED libusb_get_config_descriptor_by_value (libusb_device ∗dev, uint8_t bConfigurationValue, struct libusb_config_descriptor ∗∗config)

  Get a USB configuration descriptor with a specific bConfigurationValue.
- void API_EXPORTED libusb_free_config_descriptor (struct libusb_config_descriptor ∗config)

  Free a configuration descriptor obtained from libusb_get_active_config_descriptor() or libusb_get_config_descriptor().
- int API_EXPORTED libusb_get_ss_endpoint_companion_descriptor (struct libusb_context ∗ctx, const struct libusb_endpoint_descriptor ∗endpoint, struct libusb_ss_endpoint_companion_descriptor ∗∗ep_comp)

  Get an endpoints superspeed endpoint companion descriptor (if any)
- void API_EXPORTED libusb_free_ss_endpoint_companion_descriptor (struct libusb_ss_endpoint_companion_descriptor ∗ep_comp)

  Free a superspeed endpoint companion descriptor obtained from libusb_get_ss_endpoint_companion_descriptor().
- int API_EXPORTED libusb_get_bos_descriptor (libusb_device_handle ∗dev_handle, struct libusb_bos_descriptor ∗∗bos)

  Get a Binary Object Store (BOS) descriptor This is a BLOCKING function, which will send requests to the device.
- void API_EXPORTED libusb_free_bos_descriptor (struct libusb_bos_descriptor ∗bos)

  Free a BOS descriptor obtained from libusb_get_bos_descriptor().
- int API_EXPORTED libusb_get_usb_2_0_extension_descriptor (struct libusb_context ∗ctx, struct libusb_bos_dev_capability_descriptor ∗dev_cap, struct libusb_usb_2_0_extension_descriptor ∗∗usb_2_0_extension)

  Get an USB 2.0 Extension descriptor.
- void API_EXPORTED libusb_free_usb_2_0_extension_descriptor (struct libusb_usb_2_0_extension_descriptor ∗usb_2_0_extension)

  Free a USB 2.0 Extension descriptor obtained from libusb_get_usb_2_0_extension_descriptor().
- int API_EXPORTED libusb_get_ss_usb_device_capability_descriptor (struct libusb_context ∗ctx, struct libusb_bos_dev_capability_descriptor ∗dev_cap, struct libusb_ss_usb_device_capability_descriptor ∗∗ss_usb_device_cap)

  Get a SuperSpeed USB Device Capability descriptor.
- void API_EXPORTED libusb_free_ss_usb_device_capability_descriptor (struct libusb_ss_usb_device_capability_descriptor ∗ss_usb_device_cap)

  Free a SuperSpeed USB Device Capability descriptor obtained from libusb_get_ss_usb_device_capability_descriptor().
- int API_EXPORTED libusb_get_container_id_descriptor (struct libusb_context ∗ctx, struct libusb_bos_dev_capability_descriptor ∗dev_cap, struct libusb_container_id_descriptor ∗∗container_id)

  Get a Container ID descriptor.
- void API_EXPORTED libusb_free_container_id_descriptor (struct libusb_container_id_descriptor ∗container_id)

  Free a Container ID descriptor obtained from libusb_get_container_id_descriptor().
- int API_EXPORTED libusb_get_string_descriptor_ascii (libusb_device_handle ∗dev_handle, uint8_t desc_index, unsigned char ∗data, int length)

  Retrieve a string descriptor in C style ASCII.

34.228.1 Macro Definition Documentation

#define DESC_HEADER_LENGTH 2

#define DEVICE_DESC_LENGTH 18

#define CONFIG_DESC_LENGTH 9

#define INTERFACE_DESC_LENGTH 9

#define ENDPOINT_DESC_LENGTH 7

#define ENDPOINT_AUDIO_DESC_LENGTH 9

34.228.2 Function Documentation

int usbi_parse_descriptor (const unsigned char ∗ *source,* const char ∗ *descriptor,* void ∗ *dest,* int *host_endian*)

int usbi_device_cache_descriptor (libusb_device ∗ *dev*)

int usbi_get_config_index_by_value (struct libusb_device ∗ *dev,* uint8_t *bConfigurationValue,* int ∗ *idx*)

34.229 lib/ndkbuild/libusb/libusb/hotplug.c File Reference

```
#include <config.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "libusbi.h"
#include "hotplug.h"
```

Functions

- void usbi_hotplug_match (struct libusb_context ∗ctx, struct libusb_device ∗dev, libusb_hotplug_event event)
- void usbi_hotplug_notification (struct libusb_context ∗ctx, struct libusb_device ∗dev, libusb_hotplug_event event)
- int API_EXPORTED libusb_hotplug_register_callback (libusb_context ∗ctx, libusb_hotplug_event events, libusb_-hotplug_flag flags, int vendor_id, int product_id, int dev_class, libusb_hotplug_callback_fn cb_fn, void ∗user_data, libusb_hotplug_callback_handle ∗callback_handle)

  Register a hotplug callback function.
- void API_EXPORTED libusb_hotplug_deregister_callback (struct libusb_context ∗ctx, libusb_hotplug_callback_-handle callback_handle)

  Deregisters a hotplug callback.
- void usbi_hotplug_deregister_all (struct libusb_context ∗ctx)

34.229.1 Function Documentation

void usbi_hotplug_match (struct libusb_context ∗ *ctx,* struct libusb_device ∗ *dev,* libusb_hotplug_event *event*)

void usbi_hotplug_notification (struct libusb_context ∗ *ctx,* struct libusb_device ∗ *dev,* libusb_hotplug_event *event*)

void usbi_hotplug_deregister_all (struct libusb_context ∗ *ctx*)

34.230 lib/ndkbuild/libusb/libusb/hotplug.h File Reference

```
#include "libusbi.h"
```

Data Structures

- struct libusb_hotplug_callback

  The hotplug callback structure.
- struct libusb_hotplug_message

Typedefs

- typedef struct libusb_hotplug_callback libusb_hotplug_callback
- typedef struct libusb_hotplug_message libusb_hotplug_message

Functions

- void usbi_hotplug_deregister_all (struct libusb_context *ctx)
- void usbi_hotplug_match (struct libusb_context *ctx, struct libusb_device *dev, libusb_hotplug_event event)
- void usbi_hotplug_notification (struct libusb_context *ctx, struct libusb_device *dev, libusb_hotplug_event event)

34.230.1 Typedef Documentation

typedef struct libusb_hotplug_callback libusb_hotplug_callback

typedef struct libusb_hotplug_message libusb_hotplug_message

34.230.2 Function Documentation

void usbi_hotplug_deregister_all (struct libusb_context * *ctx*)

void usbi_hotplug_match (struct libusb_context * *ctx,* struct libusb_device * *dev,* libusb_hotplug_event *event*)

void usbi_hotplug_notification (struct libusb_context * *ctx,* struct libusb_device * *dev,* libusb_hotplug_event *event*)

34.231 lib/ndkbuild/libusb/libusb/io.c File Reference

```
#include <config.h>
#include <assert.h>
#include <errno.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "libusbi.h"
#include "hotplug.h"
```

Functions

- int usbi_io_init (struct libusb_context *ctx)
- void usbi_io_exit (struct libusb_context *ctx)
- DEFAULT_VISIBILITY struct libusb_transfer *LIBUSB_CALL libusb_alloc_transfer (int iso_packets)

  Allocate a libusb transfer with a specified number of isochronous packet descriptors.
- void API_EXPORTED libusb_free_transfer (struct libusb_transfer *transfer)

  Free a transfer structure.
- int API_EXPORTED libusb_submit_transfer (struct libusb_transfer *transfer)

  Submit a transfer.
- int API_EXPORTED libusb_cancel_transfer (struct libusb_transfer *transfer)

  Asynchronously cancel a previously submitted transfer.
- void API_EXPORTED libusb_transfer_set_stream_id (struct libusb_transfer *transfer, uint32_t stream_id)

  Set a transfers bulk stream id.
- uint32_t API_EXPORTED libusb_transfer_get_stream_id (struct libusb_transfer *transfer)

  Get a transfers bulk stream id.
- int usbi_handle_transfer_completion (struct usbi_transfer *itransfer, enum libusb_transfer_status status)
- int usbi_handle_transfer_cancellation (struct usbi_transfer *transfer)
- void usbi_signal_transfer_completion (struct usbi_transfer *transfer)
- int API_EXPORTED libusb_try_lock_events (libusb_context *ctx)

  Attempt to acquire the event handling lock.
- void API_EXPORTED libusb_lock_events (libusb_context *ctx)

  Acquire the event handling lock, blocking until successful acquisition if it is contended.
- void API_EXPORTED libusb_unlock_events (libusb_context *ctx)

Release the lock previously acquired with libusb_try_lock_events() or libusb_lock_events().

- int API_EXPORTED libusb_event_handling_ok (libusb_context *ctx)

  Determine if it is still OK for this thread to be doing event handling.
- int API_EXPORTED libusb_event_handler_active (libusb_context *ctx)

  Determine if an active thread is handling events (i.e.
- void API_EXPORTED libusb_interrupt_event_handler (libusb_context *ctx)

  Interrupt any active thread that is handling events.
- void API_EXPORTED libusb_lock_event_waiters (libusb_context *ctx)

  Acquire the event waiters lock.
- void API_EXPORTED libusb_unlock_event_waiters (libusb_context *ctx)

  Release the event waiters lock.
- int API_EXPORTED libusb_wait_for_event (libusb_context *ctx, struct timeval *tv)

  Wait for another thread to signal completion of an event.
- int API_EXPORTED libusb_handle_events_timeout_completed (libusb_context *ctx, struct timeval *tv, int *completed)

  Handle any pending events.
- int API_EXPORTED libusb_handle_events_timeout (libusb_context *ctx, struct timeval *tv)

  Handle any pending events.
- int API_EXPORTED libusb_handle_events (libusb_context *ctx)

  Handle any pending events in blocking mode.
- int API_EXPORTED libusb_handle_events_completed (libusb_context *ctx, int *completed)

  Handle any pending events in blocking mode.
- int API_EXPORTED libusb_handle_events_locked (libusb_context *ctx, struct timeval *tv)

  Handle any pending events by polling file descriptors, without checking if any other threads are already doing so.
- int API_EXPORTED libusb_pollfds_handle_timeouts (libusb_context *ctx)

  Determines whether your application must apply special timing considerations when monitoring libusb's file descriptors.
- int API_EXPORTED libusb_get_next_timeout (libusb_context *ctx, struct timeval *tv)

  Determine the next internal timeout that libusb needs to handle.
- void API_EXPORTED libusb_set_pollfd_notifiers (libusb_context *ctx, libusb_pollfd_added_cb added_cb, libusb_pollfd_removed_cb removed_cb, void *user_data)

  Register notification functions for file descriptor additions/removals.
- int usbi_add_pollfd (struct libusb_context *ctx, int fd, short events)
- void usbi_remove_pollfd (struct libusb_context *ctx, int fd)
- DEFAULT_VISIBILITY const struct libusb_pollfd **LIBUSB_CALL libusb_get_pollfds (libusb_context *ctx)

  Retrieve a list of file descriptors that should be polled by your main loop as libusb event sources.
- void API_EXPORTED libusb_free_pollfds (const struct libusb_pollfd **pollfds)

  Free a list of libusb_pollfd structures.
- void usbi_handle_disconnect (struct libusb_device_handle *dev_handle)

34.231.1 Function Documentation

int usbi_io_init (struct libusb_context * *ctx*)

void usbi_io_exit (struct libusb_context * *ctx*)

int usbi_handle_transfer_completion (struct usbi_transfer * *itransfer,* enum libusb_transfer_status *status*)

int usbi_handle_transfer_cancellation (struct usbi_transfer * *transfer*)

void usbi_signal_transfer_completion (struct usbi_transfer * *transfer*)

int usbi_add_pollfd (struct libusb_context * *ctx,* int *fd,* short *events*)

void usbi_remove_pollfd (struct libusb_context * *ctx,* int *fd*)

void usbi_handle_disconnect (struct libusb_device_handle * *dev_handle*)

34.232 lib/ndkbuild/libusb/libusb/libusb.h File Reference

```
#include <stdint.h>
#include <sys/types.h>
#include <time.h>
#include <limits.h>
```

Data Structures

- struct libusb_device_descriptor

  A structure representing the standard USB device descriptor.
- struct libusb_endpoint_descriptor

  A structure representing the standard USB endpoint descriptor.
- struct libusb_interface_descriptor

  A structure representing the standard USB interface descriptor.
- struct libusb_interface

  A collection of alternate settings for a particular USB interface.
- struct libusb_config_descriptor

  A structure representing the standard USB configuration descriptor.
- struct libusb_ss_endpoint_companion_descriptor

  A structure representing the superspeed endpoint companion descriptor.
- struct libusb_bos_dev_capability_descriptor

  A generic representation of a BOS Device Capability descriptor.
- struct libusb_bos_descriptor

  A structure representing the Binary Device Object Store (BOS) descriptor.
- struct libusb_usb_2_0_extension_descriptor

  A structure representing the USB 2.0 Extension descriptor This descriptor is documented in section 9.6.2.1 of the USB 3.0 specification.
- struct libusb_ss_usb_device_capability_descriptor

  A structure representing the SuperSpeed USB Device Capability descriptor This descriptor is documented in section 9.6.2.2 of the USB 3.0 specification.
- struct libusb_container_id_descriptor

  A structure representing the Container ID descriptor.
- struct libusb_control_setup

  Setup packet for control transfers.
- struct libusb_version

  Structure providing the version of the libusb runtime.
- struct libusb_iso_packet_descriptor

  Isochronous packet descriptor.
- struct libusb_transfer

  The generic USB transfer structure.
- struct libusb_pollfd

  File descriptor for polling.

Macros

- #define LIBUSB_DEPRECATED_FOR(f)
- #define LIBUSB_CALL

  libusb's Windows calling convention.
- #define LIBUSB_API_VERSION 0x01000105

  libusb's API version.
- #define LIBUSBX_API_VERSION LIBUSB_API_VERSION
- #define libusb_le16_to_cpu libusb_cpu_to_le16

  Convert a 16-bit value from little-endian to host-endian format.
- #define LIBUSB_DT_DEVICE_SIZE 18
- #define LIBUSB_DT_CONFIG_SIZE 9
- #define LIBUSB_DT_INTERFACE_SIZE 9
- #define LIBUSB_DT_ENDPOINT_SIZE 7
- #define LIBUSB_DT_ENDPOINT_AUDIO_SIZE 9 /∗ Audio extension ∗/
- #define LIBUSB_DT_HUB_NONVAR_SIZE 7
- #define LIBUSB_DT_SS_ENDPOINT_COMPANION_SIZE 6
- #define LIBUSB_DT_BOS_SIZE 5
- #define LIBUSB_DT_DEVICE_CAPABILITY_SIZE 3
- #define LIBUSB_BT_USB_2_0_EXTENSION_SIZE 7
- #define LIBUSB_BT_SS_USB_DEVICE_CAPABILITY_SIZE 10
- #define LIBUSB_BT_CONTAINER_ID_SIZE 20
- #define LIBUSB_DT_BOS_MAX_SIZE
- #define LIBUSB_ENDPOINT_ADDRESS_MASK 0x0f /∗ in bEndpointAddress ∗/
- #define LIBUSB_ENDPOINT_DIR_MASK 0x80
- #define LIBUSB_TRANSFER_TYPE_MASK 0x03 /∗ in bmAttributes ∗/
- #define LIBUSB_ISO_SYNC_TYPE_MASK 0x0C
- #define LIBUSB_ISO_USAGE_TYPE_MASK 0x30

- #define LIBUSB_CONTROL_SETUP_SIZE (sizeof(struct libusb_control_setup))
- #define LIBUSB_ERROR_COUNT 14
- #define LIBUSB_HOTPLUG_MATCH_ANY -1

  Wildcard matching for hotplug events.

Typedefs

- typedef struct libusb_context libusb_context

  Structure representing a libusb session.
- typedef struct libusb_device libusb_device

  Structure representing a USB device detected on the system.
- typedef struct libusb_device_handle libusb_device_handle

  Structure representing a handle on a USB device.
- typedef void(LIBUSB_CALL * libusb_pollfd_added_cb)(int fd, short events, void *user_data)

  Callback function, invoked when a new file descriptor should be added to the set of file descriptors monitored for events.
- typedef void(LIBUSB_CALL * libusb_pollfd_removed_cb)(int fd, void *user_data)

  Callback function, invoked when a file descriptor should be removed from the set of file descriptors being monitored for events.
- typedef int libusb_hotplug_callback_handle

  Callback handle.
- typedef int(LIBUSB_CALL * libusb_hotplug_callback_fn)(libusb_context *ctx, libusb_device *device, libusb_hotplug_event event, void *user_data)

  Hotplug callback function type.

Enumerations

- enum libusb_class_code {
  LIBUSB_CLASS_PER_INTERFACE = 0, LIBUSB_CLASS_AUDIO = 1, LIBUSB_CLASS_COMM = 2, LIBUSB_CLASS_HID = 3,
  LIBUSB_CLASS_PHYSICAL = 5, LIBUSB_CLASS_PRINTER = 7, LIBUSB_CLASS_PTP = 6, LIBUSB_CLASS_IMAGE = 6,
  LIBUSB_CLASS_MASS_STORAGE = 8, LIBUSB_CLASS_HUB = 9, LIBUSB_CLASS_DATA = 10, LIBUSB_CLASS_SMART_CARD = 0x0b,
  LIBUSB_CLASS_CONTENT_SECURITY = 0x0d, LIBUSB_CLASS_VIDEO = 0x0e, LIBUSB_CLASS_PERSONAL_HEALTHCARE = 0x0f, LIBUSB_CLASS_DIAGNOSTIC_DEVICE = 0xdc,
  LIBUSB_CLASS_WIRELESS = 0xe0, LIBUSB_CLASS_APPLICATION = 0xfe, LIBUSB_CLASS_VENDOR_SPEC = 0xff }

  Device and/or Interface Class codes.
- enum libusb_descriptor_type {
  LIBUSB_DT_DEVICE = 0x01, LIBUSB_DT_CONFIG = 0x02, LIBUSB_DT_STRING = 0x03, LIBUSB_DT_INTERFACE = 0x04,
  LIBUSB_DT_ENDPOINT = 0x05, LIBUSB_DT_BOS = 0x0f, LIBUSB_DT_DEVICE_CAPABILITY = 0x10, LIBUSB_DT_HID = 0x21,
  LIBUSB_DT_REPORT = 0x22, LIBUSB_DT_PHYSICAL = 0x23, LIBUSB_DT_HUB = 0x29, LIBUSB_DT_SUPERSPEED_HUB = 0x2a,
  LIBUSB_DT_SS_ENDPOINT_COMPANION = 0x30 }

  Descriptor types as defined by the USB specification.
- enum libusb_endpoint_direction { LIBUSB_ENDPOINT_IN = 0x80, LIBUSB_ENDPOINT_OUT = 0x00 }

  Endpoint direction.
- enum libusb_transfer_type {
  LIBUSB_TRANSFER_TYPE_CONTROL = 0, LIBUSB_TRANSFER_TYPE_ISOCHRONOUS = 1, LIBUSB_TRANSFER_TYPE_BULK = 2, LIBUSB_TRANSFER_TYPE_INTERRUPT = 3,
  LIBUSB_TRANSFER_TYPE_BULK_STREAM = 4 }

  Endpoint transfer type.
- enum libusb_standard_request {
  LIBUSB_REQUEST_GET_STATUS = 0x00, LIBUSB_REQUEST_CLEAR_FEATURE = 0x01, LIBUSB_REQUEST_SET_FEATURE = 0x03, LIBUSB_REQUEST_SET_ADDRESS = 0x05,
  LIBUSB_REQUEST_GET_DESCRIPTOR = 0x06, LIBUSB_REQUEST_SET_DESCRIPTOR = 0x07, LIBUSB_REQUEST_GET_CONFIGURATION = 0x08, LIBUSB_REQUEST_SET_CONFIGURATION = 0x09,
  LIBUSB_REQUEST_GET_INTERFACE = 0x0A, LIBUSB_REQUEST_SET_INTERFACE = 0x0B, LIBUSB_REQUEST_SYNCH_FRAME = 0x0C, LIBUSB_REQUEST_SET_SEL = 0x30,
  LIBUSB_SET_ISOCH_DELAY = 0x31 }

  Standard requests, as defined in table 9-5 of the USB 3.0 specifications.
- enum libusb_request_type { LIBUSB_REQUEST_TYPE_STANDARD = (0x00 << 5), LIBUSB_REQUEST_TYPE_CLASS = (0x01 << 5), LIBUSB_REQUEST_TYPE_VENDOR = (0x02 << 5), LIBUSB_REQUEST_TYPE_RESERVED = (0x03 << 5) }

Request type bits of the bmRequestType field in control transfers.

- enum libusb_request_recipient { LIBUSB_RECIPIENT_DEVICE = 0x00, LIBUSB_RECIPIENT_INTERFACE = 0x01, LIBUSB_RECIPIENT_ENDPOINT = 0x02, LIBUSB_RECIPIENT_OTHER = 0x03 }

  Recipient bits of the bmRequestType field in control transfers.
- enum libusb_iso_sync_type { LIBUSB_ISO_SYNC_TYPE_NONE = 0, LIBUSB_ISO_SYNC_TYPE_ASYNC = 1, LIBUSB_ISO_SYNC_TYPE_ADAPTIVE = 2, LIBUSB_ISO_SYNC_TYPE_SYNC = 3 }

  Synchronization type for isochronous endpoints.
- enum libusb_iso_usage_type { LIBUSB_ISO_USAGE_TYPE_DATA = 0, LIBUSB_ISO_USAGE_TYPE_FEEDBACK = 1, LIBUSB_ISO_USAGE_TYPE_IMPLICIT = 2 }

  Usage type for isochronous endpoints.
- enum libusb_speed {
  LIBUSB_SPEED_UNKNOWN = 0, LIBUSB_SPEED_LOW = 1, LIBUSB_SPEED_FULL = 2, LIBUSB_SPEED_HIGH = 3,
  LIBUSB_SPEED_SUPER = 4 }

  Speed codes.
- enum libusb_supported_speed { LIBUSB_LOW_SPEED_OPERATION = 1, LIBUSB_FULL_SPEED_OPERATION = 2, LIBUSB_HIGH_SPEED_OPERATION = 4, LIBUSB_SUPER_SPEED_OPERATION = 8 }

  Supported speeds (wSpeedSupported) bitfield.
- enum libusb_usb_2_0_extension_attributes { LIBUSB_BM_LPM_SUPPORT = 2 }

  Masks for the bits of the bmAttributes field of the USB 2.0 Extension descriptor.
- enum libusb_ss_usb_device_capability_attributes { LIBUSB_BM_LTM_SUPPORT = 2 }

  Masks for the bits of the bmAttributes field field of the SuperSpeed USB Device Capability descriptor.
- enum libusb_bos_type { LIBUSB_BT_WIRELESS_USB_DEVICE_CAPABILITY = 1, LIBUSB_BT_USB_2_0_EXTENSION = 2, LIBUSB_BT_SS_USB_DEVICE_CAPABILITY = 3, LIBUSB_BT_CONTAINER_ID = 4 }

  USB capability types.
- enum libusb_error {
  LIBUSB_SUCCESS = 0, LIBUSB_ERROR_IO = -1, LIBUSB_ERROR_INVALID_PARAM = -2, LIBUSB_ERROR_ACCESS = -3,
  LIBUSB_ERROR_NO_DEVICE = -4, LIBUSB_ERROR_NOT_FOUND = -5, LIBUSB_ERROR_BUSY = -6, LIBUSB_ERROR_TIMEOUT = -7,
  LIBUSB_ERROR_OVERFLOW = -8, LIBUSB_ERROR_PIPE = -9, LIBUSB_ERROR_INTERRUPTED = -10, LIBUSB_ERROR_NO_MEM = -11,
  LIBUSB_ERROR_NOT_SUPPORTED = -12, LIBUSB_ERROR_OTHER = -99 }

  Error codes.
- enum libusb_transfer_status {
  LIBUSB_TRANSFER_COMPLETED, LIBUSB_TRANSFER_ERROR, LIBUSB_TRANSFER_TIMED_OUT, LIBUSB_TRANSFER_CANCELLED,
  LIBUSB_TRANSFER_STALL, LIBUSB_TRANSFER_NO_DEVICE, LIBUSB_TRANSFER_OVERFLOW }

  Transfer status codes.
- enum libusb_transfer_flags { LIBUSB_TRANSFER_SHORT_NOT_OK = 1<<0, LIBUSB_TRANSFER_FREE_BUFFER = 1<<1, LIBUSB_TRANSFER_FREE_TRANSFER = 1<<2, LIBUSB_TRANSFER_ADD_ZERO_PACKET = 1 << 3 }

  libusb_transfer.flags values
- enum libusb_capability { LIBUSB_CAP_HAS_CAPABILITY = 0x0000, LIBUSB_CAP_HAS_HOTPLUG = 0x0001, LIBUSB_CAP_HAS_HID_ACCESS = 0x0100, LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER = 0x0101 }

  Capabilities supported by an instance of libusb on the current running platform.
- enum libusb_log_level {
  LIBUSB_LOG_LEVEL_NONE = 0, LIBUSB_LOG_LEVEL_ERROR, LIBUSB_LOG_LEVEL_WARNING, LIBUSB_LOG_LEVEL_INFO,
  LIBUSB_LOG_LEVEL_DEBUG }

  Log message levels.
- enum libusb_hotplug_flag { LIBUSB_HOTPLUG_NO_FLAGS = 0, LIBUSB_HOTPLUG_ENUMERATE = 1<<0 }

  Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.
- enum libusb_hotplug_event { LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED = 0x01, LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT = 0x02 }

  Since version 1.0.16, LIBUSB_API_VERSION >= 0x01000102.

Functions

- typedef void (LIBUSB_CALL *libusb_transfer_cb_fn)(struct libusb_transfer *transfer)

  Asynchronous transfer callback function type.
- int LIBUSB_CALL libusb_init (libusb_context **ctx)

  Initialize libusb.
- void LIBUSB_CALL libusb_exit (libusb_context *ctx)

  Deinitialize libusb.

- void LIBUSB_CALL libusb_set_debug (libusb_context *ctx, int level)

  Set log message verbosity.
- const struct libusb_version *LIBUSB_CALL libusb_get_version (void)

  Returns a pointer to const struct libusb_version with the version (major, minor, micro, nano and rc) of the running library.
- int LIBUSB_CALL libusb_has_capability (uint32_t capability)

  Check at runtime if the loaded library has a given capability.
- const char *LIBUSB_CALL libusb_error_name (int errcode)

  Returns a constant NULL-terminated string with the ASCII name of a libusb error or transfer status code.
- int LIBUSB_CALL libusb_setlocale (const char *locale)

  Set the language, and only the language, not the encoding! used for translatable libusb messages.
- const char *LIBUSB_CALL libusb_strerror (enum libusb_error errcode)

  Returns a constant string with a short description of the given error code, this description is intended for displaying to the end user and will be in the language set by libusb_setlocale().
- ssize_t LIBUSB_CALL libusb_get_device_list (libusb_context *ctx, libusb_device ***list)

  Returns a list of USB devices currently attached to the system.
- void LIBUSB_CALL libusb_free_device_list (libusb_device **list, int unref_devices)

  Frees a list of devices previously discovered using libusb_get_device_list().
- libusb_device *LIBUSB_CALL libusb_ref_device (libusb_device *dev)

  Increment the reference count of a device.
- void LIBUSB_CALL libusb_unref_device (libusb_device *dev)

  Decrement the reference count of a device.
- int LIBUSB_CALL libusb_get_configuration (libusb_device_handle *dev, int *config)

  Determine the bConfigurationValue of the currently active configuration.
- int LIBUSB_CALL libusb_get_device_descriptor (libusb_device *dev, struct libusb_device_descriptor *desc)

  Get the USB device descriptor for a given device.
- int LIBUSB_CALL libusb_get_active_config_descriptor (libusb_device *dev, struct libusb_config_descriptor **config)

  Get the USB configuration descriptor for the currently active configuration.
- int LIBUSB_CALL libusb_get_config_descriptor (libusb_device *dev, uint8_t config_index, struct libusb_config_descriptor **config)

  Get a USB configuration descriptor based on its index.
- int LIBUSB_CALL libusb_get_config_descriptor_by_value (libusb_device *dev, uint8_t bConfigurationValue, struct libusb_config_descriptor **config)

  Get a USB configuration descriptor with a specific bConfigurationValue.
- void LIBUSB_CALL libusb_free_config_descriptor (struct libusb_config_descriptor *config)

  Free a configuration descriptor obtained from libusb_get_active_config_descriptor() or libusb_get_config_descriptor().
- int LIBUSB_CALL libusb_get_ss_endpoint_companion_descriptor (struct libusb_context *ctx, const struct libusb_endpoint_descriptor *endpoint, struct libusb_ss_endpoint_companion_descriptor **ep_comp)

  Get an endpoints superspeed endpoint companion descriptor (if any)
- void LIBUSB_CALL libusb_free_ss_endpoint_companion_descriptor (struct libusb_ss_endpoint_companion_descriptor *ep_comp)

  Free a superspeed endpoint companion descriptor obtained from libusb_get_ss_endpoint_companion_descriptor().
- int LIBUSB_CALL libusb_get_bos_descriptor (libusb_device_handle *dev_handle, struct libusb_bos_descriptor **bos)

  Get a Binary Object Store (BOS) descriptor This is a BLOCKING function, which will send requests to the device.
- void LIBUSB_CALL libusb_free_bos_descriptor (struct libusb_bos_descriptor *bos)

  Free a BOS descriptor obtained from libusb_get_bos_descriptor().
- int LIBUSB_CALL libusb_get_usb_2_0_extension_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_usb_2_0_extension_descriptor **usb_2_0_extension)

  Get an USB 2.0 Extension descriptor.
- void LIBUSB_CALL libusb_free_usb_2_0_extension_descriptor (struct libusb_usb_2_0_extension_descriptor *usb_2_0_extension)

  Free a USB 2.0 Extension descriptor obtained from libusb_get_usb_2_0_extension_descriptor().
- int LIBUSB_CALL libusb_get_ss_usb_device_capability_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_ss_usb_device_capability_descriptor **ss_usb_device_cap)

  Get a SuperSpeed USB Device Capability descriptor.
- void LIBUSB_CALL libusb_free_ss_usb_device_capability_descriptor (struct libusb_ss_usb_device_capability_descriptor *ss_usb_device_cap)

  Free a SuperSpeed USB Device Capability descriptor obtained from libusb_get_ss_usb_device_capability_descriptor().
- int LIBUSB_CALL libusb_get_container_id_descriptor (struct libusb_context *ctx, struct libusb_bos_dev_capability_descriptor *dev_cap, struct libusb_container_id_descriptor **container_id)

  Get a Container ID descriptor.
- void LIBUSB_CALL libusb_free_container_id_descriptor (struct libusb_container_id_descriptor *container_id)

Free a Container ID descriptor obtained from libusb_get_container_id_descriptor().

- uint8_t LIBUSB_CALL libusb_get_bus_number (libusb_device *dev)

  Get the number of the bus that a device is connected to.
- uint8_t LIBUSB_CALL libusb_get_port_number (libusb_device *dev)

  Get the number of the port that a device is connected to.
- int LIBUSB_CALL libusb_get_port_numbers (libusb_device *dev, uint8_t *port_numbers, int port_numbers_len)

  Get the list of all port numbers from root for the specified device.
- int LIBUSB_CALL libusb_get_port_path (libusb_context *ctx, libusb_device *dev, uint8_t *path, uint8_t path_length)

  Deprecated please use libusb_get_port_numbers instead.
- libusb_device *LIBUSB_CALL libusb_get_parent (libusb_device *dev)

  Get the the parent from the specified device.
- uint8_t LIBUSB_CALL libusb_get_device_address (libusb_device *dev)

  Get the address of the device on the bus it is connected to.
- int LIBUSB_CALL libusb_get_device_speed (libusb_device *dev)

  Get the negotiated connection speed for a device.
- int LIBUSB_CALL libusb_get_max_packet_size (libusb_device *dev, unsigned char endpoint)

  Convenience function to retrieve the wMaxPacketSize value for a particular endpoint in the active device configuration.
- int LIBUSB_CALL libusb_get_max_iso_packet_size (libusb_device *dev, unsigned char endpoint)

  Calculate the maximum packet size which a specific endpoint is capable is sending or receiving in the duration of 1 microframe.
- int LIBUSB_CALL libusb_open (libusb_device *dev, libusb_device_handle **dev_handle)

  Open a device and obtain a device handle.
- void LIBUSB_CALL libusb_close (libusb_device_handle *dev_handle)

  Close a device handle.
- libusb_device *LIBUSB_CALL libusb_get_device (libusb_device_handle *dev_handle)

  Get the underlying device for a device handle.
- int LIBUSB_CALL libusb_set_configuration (libusb_device_handle *dev_handle, int configuration)

  Set the active configuration for a device.
- int LIBUSB_CALL libusb_claim_interface (libusb_device_handle *dev_handle, int interface_number)

  Claim an interface on a given device handle.
- int LIBUSB_CALL libusb_release_interface (libusb_device_handle *dev_handle, int interface_number)

  Release an interface previously claimed with libusb_claim_interface().
- libusb_device_handle *LIBUSB_CALL libusb_open_device_with_vid_pid (libusb_context *ctx, uint16_t vendor_id, uint16_t product_id)

  Convenience function for finding a device with a particular `idVendor/idProduct` combination.
- int LIBUSB_CALL libusb_set_interface_alt_setting (libusb_device_handle *dev_handle, int interface_number, int alternate_setting)

  Activate an alternate setting for an interface.
- int LIBUSB_CALL libusb_clear_halt (libusb_device_handle *dev_handle, unsigned char endpoint)

  Clear the halt/stall condition for an endpoint.
- int LIBUSB_CALL libusb_reset_device (libusb_device_handle *dev_handle)

  Perform a USB port reset to reinitialize a device.
- int LIBUSB_CALL libusb_alloc_streams (libusb_device_handle *dev_handle, uint32_t num_streams, unsigned char *endpoints, int num_endpoints)

  Allocate up to num_streams usb bulk streams on the specified endpoints.
- int LIBUSB_CALL libusb_free_streams (libusb_device_handle *dev_handle, unsigned char *endpoints, int num_endpoints)

  Free usb bulk streams allocated with libusb_alloc_streams().
- unsigned char *LIBUSB_CALL libusb_dev_mem_alloc (libusb_device_handle *dev_handle, size_t length)

  Attempts to allocate a block of persistent DMA memory suitable for transfers against the given device.
- int LIBUSB_CALL libusb_dev_mem_free (libusb_device_handle *dev_handle, unsigned char *buffer, size_t length)

  Free device memory allocated with libusb_dev_mem_alloc().
- int LIBUSB_CALL libusb_kernel_driver_active (libusb_device_handle *dev_handle, int interface_number)

  Determine if a kernel driver is active on an interface.
- int LIBUSB_CALL libusb_detach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Detach a kernel driver from an interface.
- int LIBUSB_CALL libusb_attach_kernel_driver (libusb_device_handle *dev_handle, int interface_number)

  Re-attach an interface's kernel driver, which was previously detached using libusb_detach_kernel_driver().
- int LIBUSB_CALL libusb_set_auto_detach_kernel_driver (libusb_device_handle *dev_handle, int enable)

  Enable/disable libusb's automatic kernel driver detachment.
- struct libusb_transfer *LIBUSB_CALL libusb_alloc_transfer (int iso_packets)

  Allocate a libusb transfer with a specified number of isochronous packet descriptors.

- int LIBUSB_CALL libusb_submit_transfer (struct libusb_transfer *transfer)

  Submit a transfer.
- int LIBUSB_CALL libusb_cancel_transfer (struct libusb_transfer *transfer)

  Asynchronously cancel a previously submitted transfer.
- void LIBUSB_CALL libusb_free_transfer (struct libusb_transfer *transfer)

  Free a transfer structure.
- void LIBUSB_CALL libusb_transfer_set_stream_id (struct libusb_transfer *transfer, uint32_t stream_id)

  Set a transfers bulk stream id.
- uint32_t LIBUSB_CALL libusb_transfer_get_stream_id (struct libusb_transfer *transfer)

  Get a transfers bulk stream id.
- int LIBUSB_CALL libusb_control_transfer (libusb_device_handle *dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char *data, uint16_t wLength, unsigned int timeout)

  Perform a USB control transfer.
- int LIBUSB_CALL libusb_bulk_transfer (libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *actual_length, unsigned int timeout)

  Perform a USB bulk transfer.
- int LIBUSB_CALL libusb_interrupt_transfer (libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *actual_length, unsigned int timeout)

  Perform a USB interrupt transfer.
- int LIBUSB_CALL libusb_get_string_descriptor_ascii (libusb_device_handle *dev_handle, uint8_t desc_index, unsigned char *data, int length)

  Retrieve a string descriptor in C style ASCII.
- int LIBUSB_CALL libusb_try_lock_events (libusb_context *ctx)

  Attempt to acquire the event handling lock.
- void LIBUSB_CALL libusb_lock_events (libusb_context *ctx)

  Acquire the event handling lock, blocking until successful acquisition if it is contended.
- void LIBUSB_CALL libusb_unlock_events (libusb_context *ctx)

  Release the lock previously acquired with libusb_try_lock_events() or libusb_lock_events().
- int LIBUSB_CALL libusb_event_handling_ok (libusb_context *ctx)

  Determine if it is still OK for this thread to be doing event handling.
- int LIBUSB_CALL libusb_event_handler_active (libusb_context *ctx)

  Determine if an active thread is handling events (i.e.
- void LIBUSB_CALL libusb_interrupt_event_handler (libusb_context *ctx)

  Interrupt any active thread that is handling events.
- void LIBUSB_CALL libusb_lock_event_waiters (libusb_context *ctx)

  Acquire the event waiters lock.
- void LIBUSB_CALL libusb_unlock_event_waiters (libusb_context *ctx)

  Release the event waiters lock.
- int LIBUSB_CALL libusb_wait_for_event (libusb_context *ctx, struct timeval *tv)

  Wait for another thread to signal completion of an event.
- int LIBUSB_CALL libusb_handle_events_timeout (libusb_context *ctx, struct timeval *tv)

  Handle any pending events.
- int LIBUSB_CALL libusb_handle_events_timeout_completed (libusb_context *ctx, struct timeval *tv, int *completed)

  Handle any pending events.
- int LIBUSB_CALL libusb_handle_events (libusb_context *ctx)

  Handle any pending events in blocking mode.
- int LIBUSB_CALL libusb_handle_events_completed (libusb_context *ctx, int *completed)

  Handle any pending events in blocking mode.
- int LIBUSB_CALL libusb_handle_events_locked (libusb_context *ctx, struct timeval *tv)

  Handle any pending events by polling file descriptors, without checking if any other threads are already doing so.
- int LIBUSB_CALL libusb_pollfds_handle_timeouts (libusb_context *ctx)

  Determines whether your application must apply special timing considerations when monitoring libusb's file descriptors.
- int LIBUSB_CALL libusb_get_next_timeout (libusb_context *ctx, struct timeval *tv)

  Determine the next internal timeout that libusb needs to handle.
- const struct libusb_pollfd **LIBUSB_CALL libusb_get_pollfds (libusb_context *ctx)

  Retrieve a list of file descriptors that should be polled by your main loop as libusb event sources.
- void LIBUSB_CALL libusb_free_pollfds (const struct libusb_pollfd **pollfds)

  Free a list of libusb_pollfd structures.
- void LIBUSB_CALL libusb_set_pollfd_notifiers (libusb_context *ctx, libusb_pollfd_added_cb added_cb, libusb_pollfd_removed_cb removed_cb, void *user_data)

  Register notification functions for file descriptor additions/removals.

- int LIBUSB_CALL libusb_hotplug_register_callback (libusb_context *ctx, libusb_hotplug_event events, libusb_hotplug_flag flags, int vendor_id, int product_id, int dev_class, libusb_hotplug_callback_fn cb_fn, void *user_data, libusb_hotplug_callback_handle *callback_handle)

  Register a hotplug callback function.

- void LIBUSB_CALL libusb_hotplug_deregister_callback (libusb_context *ctx, libusb_hotplug_callback_handle callback_handle)

  Deregisters a hotplug callback.

34.232.1 Macro Definition Documentation

#define LIBUSB_DEPRECATED_FOR(*f*)

#define LIBUSBX_API_VERSION LIBUSB_API_VERSION

#define LIBUSB_DT_DEVICE_SIZE 18

#define LIBUSB_DT_CONFIG_SIZE 9

#define LIBUSB_DT_INTERFACE_SIZE 9

#define LIBUSB_DT_ENDPOINT_SIZE 7

#define LIBUSB_DT_ENDPOINT_AUDIO_SIZE 9 /* Audio extension */

#define LIBUSB_DT_HUB_NONVAR_SIZE 7

#define LIBUSB_DT_SS_ENDPOINT_COMPANION_SIZE 6

#define LIBUSB_DT_BOS_SIZE 5

#define LIBUSB_DT_DEVICE_CAPABILITY_SIZE 3

#define LIBUSB_BT_USB_2_0_EXTENSION_SIZE 7

#define LIBUSB_BT_SS_USB_DEVICE_CAPABILITY_SIZE 10

#define LIBUSB_BT_CONTAINER_ID_SIZE 20

#define LIBUSB_DT_BOS_MAX_SIZE

Value:

```
((LIBUSB_DT_BOS_SIZE)      +\
                                        (LIBUSB_BT_USB_2_0_EXTENSION_SIZE)
            +\
                                        (LIBUSB_BT_SS_USB_DEVICE_CAPABILITY_SIZE
      ) +\
                                        (LIBUSB_BT_CONTAINER_ID_SIZE))
```

#define LIBUSB_ENDPOINT_ADDRESS_MASK 0x0f /* in bEndpointAddress */

#define LIBUSB_ENDPOINT_DIR_MASK 0x80

#define LIBUSB_TRANSFER_TYPE_MASK 0x03 /* in bmAttributes */

#define LIBUSB_ISO_SYNC_TYPE_MASK 0x0C

#define LIBUSB_ISO_USAGE_TYPE_MASK 0x30

#define LIBUSB_CONTROL_SETUP_SIZE (sizeof(struct libusb_control_setup))

#define LIBUSB_ERROR_COUNT 14

34.233 lib/ndkbuild/libusb/libusb/libusbi.h File Reference

```
#include <config.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <time.h>
#include <stdarg.h>
#include "libusb.h"
#include "version.h"
```

Data Structures

- struct list_head
- struct libusb_context
- struct libusb_device
- struct libusb_device_handle
- struct usbi_transfer
- struct usb_descriptor_header
- struct usbi_pollfd
- struct discovered_devs
- struct usbi_os_backend

Macros

- #define API_EXPORTED LIBUSB_CALL DEFAULT_VISIBILITY
- #define DEVICE_DESC_LENGTH 18
- #define USB_MAXENDPOINTS 32
- #define USB_MAXINTERFACES 32
- #define USB_MAXCONFIG 8
- #define USBI_CAP_HAS_HID_ACCESS 0x00010000
- #define USBI_CAP_SUPPORTS_DETACH_KERNEL_DRIVER 0x00020000
- #define USBI_MAX_LOG_LEN 1024
- #define USBI_LOG_LINE_END "\n"
- #define UNUSED(var) do { (void)(var); } while(0)
- #define ARRAYSIZE(array) (sizeof(array) / sizeof(array[0]))
- #define list_entry(ptr, type, member) ((type *)((uintptr_t)(ptr) - (uintptr_t)offsetof(type, member)))
- #define list_first_entry(ptr, type, member) list_entry((ptr)->next, type, member)
- #define list_for_each_entry(pos, head, member, type)
- #define list_for_each_entry_safe(pos, n, head, member, type)
- #define list_empty(entry) ((entry)->next == (entry))
- #define container_of(ptr, type, member)
- #define MIN(a, b) ((a) < (b) ? (a) : (b))
- #define MAX(a, b) ((a) > (b) ? (a) : (b))
- #define TIMESPEC_IS_SET(ts) ((ts)->tv_sec != 0 || (ts)->tv_nsec != 0)
- #define TIMEVAL_TV_SEC_TYPE time_t
- #define TIMESPEC_TO_TIMEVAL(tv, ts)
- #define usbi_dbg(...) do {} while(0)
- #define usbi_info(ctx,...) _usbi_log(ctx, LIBUSB_LOG_LEVEL_INFO, __VA_ARGS__)
- #define usbi_warn(ctx,...) _usbi_log(ctx, LIBUSB_LOG_LEVEL_WARNING, __VA_ARGS__)
- #define usbi_err(ctx,...) _usbi_log(ctx, LIBUSB_LOG_LEVEL_ERROR, __VA_ARGS__)
- #define USBI_GET_CONTEXT(ctx)
- #define DEVICE_CTX(dev) ((dev)->ctx)
- #define HANDLE_CTX(handle) (DEVICE_CTX((handle)->dev))
- #define TRANSFER_CTX(transfer) (HANDLE_CTX((transfer)->dev_handle))
- #define ITRANSFER_CTX(transfer) (TRANSFER_CTX(USBI_TRANSFER_TO_LIBUSB_TRANSFER(transfer)))
- #define IS_EPIN(ep) (0 != ((ep) & LIBUSB_ENDPOINT_IN))
- #define IS_EPOUT(ep) (!IS_EPIN(ep))
- #define IS_XFERIN(xfer) (0 != ((xfer)->endpoint & LIBUSB_ENDPOINT_IN))
- #define IS_XFEROUT(xfer) (!IS_XFERIN(xfer))
- #define usbi_handling_events(ctx) (usbi_tls_key_get((ctx)->event_handling_key) != NULL)
- #define usbi_start_event_handling(ctx) usbi_tls_key_set((ctx)->event_handling_key, ctx)
- #define usbi_end_event_handling(ctx) usbi_tls_key_set((ctx)->event_handling_key, NULL)
- #define usbi_pending_events(ctx)
- #define usbi_using_timerfd(ctx) (0)
- #define USBI_TRANSFER_TO_LIBUSB_TRANSFER(transfer)
- #define LIBUSB_TRANSFER_TO_USBI_TRANSFER(transfer)

Enumerations

- enum usbi_event_flags { USBI_EVENT_POLLFDS_MODIFIED = 1 << 0, USBI_EVENT_USER_INTERRUPT = 1 << 1 }
- enum { USBI_CLOCK_MONOTONIC, USBI_CLOCK_REALTIME }
- enum usbi_transfer_state_flags { USBI_TRANSFER_IN_FLIGHT = 1 << 0, USBI_TRANSFER_CANCELLING = 1 << 1, USBI_TRANSFER_DEVICE_DISAPPEARED = 1 << 2 }
- enum usbi_transfer_timeout_flags { USBI_TRANSFER_OS_HANDLES_TIMEOUT = 1 << 0, USBI_TRANSFER_TIMEOUT_HANDLED = 1 << 1, USBI_TRANSFER_TIMED_OUT = 1 << 2 }

Functions

- void usbi_log (struct libusb_context *ctx, enum libusb_log_level level, const char *function, const char *format,...)
- void usbi_log_v (struct libusb_context *ctx, enum libusb_log_level level, const char *function, const char *format, va_list args)
- int usbi_io_init (struct libusb_context *ctx)
- void usbi_io_exit (struct libusb_context *ctx)
- struct libusb_device * usbi_alloc_device (struct libusb_context *ctx, unsigned long session_id)
- struct libusb_device * usbi_get_device_by_session_id (struct libusb_context *ctx, unsigned long session_id)
- int usbi_sanitize_device (struct libusb_device *dev)
- void usbi_handle_disconnect (struct libusb_device_handle *dev_handle)
- int usbi_handle_transfer_completion (struct usbi_transfer *itransfer, enum libusb_transfer_status status)
- int usbi_handle_transfer_cancellation (struct usbi_transfer *transfer)
- void usbi_signal_transfer_completion (struct usbi_transfer *transfer)
- int usbi_parse_descriptor (const unsigned char *source, const char *descriptor, void *dest, int host_endian)
- int usbi_device_cache_descriptor (libusb_device *dev)
- int usbi_get_config_index_by_value (struct libusb_device *dev, uint8_t bConfigurationValue, int *idx)
- void usbi_connect_device (struct libusb_device *dev)
- void usbi_disconnect_device (struct libusb_device *dev)
- int usbi_signal_event (struct libusb_context *ctx)
- int usbi_clear_event (struct libusb_context *ctx)
- int usbi_add_pollfd (struct libusb_context *ctx, int fd, short events)
- void usbi_remove_pollfd (struct libusb_context *ctx, int fd)
- struct discovered_devs * discovered_devs_append (struct discovered_devs *discdevs, struct libusb_device *dev)

Variables

- struct libusb_context * usbi_default_context
- const struct usbi_os_backend *const usbi_backend
- const struct usbi_os_backend linux_usbfs_backend
- const struct usbi_os_backend darwin_backend
- const struct usbi_os_backend openbsd_backend
- const struct usbi_os_backend netbsd_backend
- const struct usbi_os_backend windows_backend
- const struct usbi_os_backend usbdk_backend
- const struct usbi_os_backend wince_backend
- const struct usbi_os_backend haiku_usb_raw_backend
- const struct usbi_os_backend sunos_backend
- struct list_head active_contexts_list
- usbi_mutex_static_t active_contexts_lock

34.233.1 Macro Definition Documentation

#define API_EXPORTED LIBUSB_CALL DEFAULT_VISIBILITY

#define DEVICE_DESC_LENGTH 18

#define USB_MAXENDPOINTS 32

#define USB_MAXINTERFACES 32

#define USB_MAXCONFIG 8

#define USBI_CAP_HAS_HID_ACCESS 0x00010000

#define USBI_CAP_SUPPORTS_DETACH_KERNEL_DRIVER 0x00020000

#define USBI_MAX_LOG_LEN 1024

#define USBI_LOG_LINE_END "\n"

#define UNUSED(*var*) do { (void)(var); } while(0)

#define ARRAYSIZE(*array*) (sizeof(array) / sizeof(array[0]))

#define list_entry(*ptr, type, member*) ((type *)((uintptr_t)(ptr) - (uintptr_t)offsetof(type, member)))

#define list_first_entry(*ptr, type, member*) list_entry((ptr)->next, type, member)

#define list_for_each_entry(*pos, head, member, type*)

Value:

```
for (pos = list_entry((head)->next, type, member);            \
                &pos->member != (head);                                \
                pos = list_entry(pos->member.next, type, member))
```

#define list_for_each_entry_safe(*pos, n, head, member, type*)

Value:

```
for (pos = list_entry((head)->next, type, member),            \
                n = list_entry(pos->member.next, type, member);      \
                &pos->member != (head);                                \
                pos = n, n = list_entry(n->member.next, type, member))
```

#define list_empty(*entry*) ((entry)->next == (entry))

#define container_of(*ptr, type, member*)

Value:

```
({                      \
        const typeof( ((type *)0)->member ) *mptr = (ptr);        \
        (type *)( (char *)mptr - offsetof(type,member) );})
```

#define MIN(*a, b*) ((a) < (b) ? (a) : (b))

#define MAX(*a, b*) ((a) > (b) ? (a) : (b))

#define TIMESPEC_IS_SET(*ts*) ((ts)->tv_sec != 0 || (ts)->tv_nsec != 0)

#define TIMEVAL_TV_SEC_TYPE time_t

#define TIMESPEC_TO_TIMEVAL(*tv, ts*)

Value:

```
do {                                                          \
                (tv)->tv_sec = (TIMEVAL_TV_SEC_TYPE) (ts)->
      tv_sec;   \
                (tv)->tv_usec = (ts)->tv_nsec / 1000;                  \
        } while (0)
```

#define usbi_dbg(*...*) do {} while(0)

#define usbi_info(*ctx, ...*) _usbi_log(ctx, LIBUSB_LOG_LEVEL_INFO, __VA_ARGS__)

#define usbi_warn(*ctx, ...*) _usbi_log(ctx, LIBUSB_LOG_LEVEL_WARNING, __VA_ARGS__)

#define usbi_err(*ctx, ...*) _usbi_log(ctx, LIBUSB_LOG_LEVEL_ERROR, __VA_ARGS__)

#define USBI_GET_CONTEXT(*ctx*)

Value:

```
do {                                          \
                if (!(ctx))                           \
                        (ctx) = usbi_default_context;       \
        } while(0)
```

#define DEVICE_CTX(*dev*) ((dev)->ctx)

#define HANDLE_CTX(handle) (DEVICE_CTX((handle)->dev))

#define TRANSFER_CTX(*transfer*) (HANDLE_CTX((transfer)->dev_handle))

#define ITRANSFER_CTX(*transfer*) (TRANSFER_CTX(USBI_TRANSFER_TO_LIBUSB_TRANSFER(transfer)))

#define IS_EPIN(*ep*) (0 != ((ep) & LIBUSB_ENDPOINT_IN))

#define IS_EPOUT(*ep*) (!IS_EPIN(ep))

#define IS_XFERIN(*xfer*) (0 != ((xfer)->endpoint & LIBUSB_ENDPOINT_IN))

#define IS_XFEROUT(*xfer*) (!IS_XFERIN(xfer))

#define usbi_handling_events(*ctx*) (usbi_tls_key_get((ctx)->event_handling_key) != NULL)

#define usbi_start_event_handling(*ctx*) usbi_tls_key_set((ctx)->event_handling_key, ctx)

#define usbi_end_event_handling(*ctx*) usbi_tls_key_set((ctx)->event_handling_key, NULL)

#define usbi_pending_events(*ctx*)

Value:

```
((ctx)->event_flags || (ctx)->device_close \
         || !list_empty(&(ctx)->hotplug_msgs) || !list_empty(&(ctx)->
      completed_transfers))
```

#define usbi_using_timerfd(*ctx*) (0)

#define USBI_TRANSFER_TO_LIBUSB_TRANSFER(*transfer*)

Value:

```
((struct libusb_transfer *)(((unsigned char *)(transfer))        \
                + sizeof(struct usbi_transfer)))
```

#define LIBUSB_TRANSFER_TO_USBI_TRANSFER(*transfer*)

Value:

```
((struct usbi_transfer *)(((unsigned char *)(transfer))            \
                - sizeof(struct usbi_transfer)))
```

34.233.2 Enumeration Type Documentation

enum usbi_event_flags

Enumerator

USBI_EVENT_POLLFDS_MODIFIED
USBI_EVENT_USER_INTERRUPT

anonymous enum

Enumerator

USBI_CLOCK_MONOTONIC
USBI_CLOCK_REALTIME

enum usbi_transfer_state_flags

Enumerator

USBI_TRANSFER_IN_FLIGHT
USBI_TRANSFER_CANCELLING
USBI_TRANSFER_DEVICE_DISAPPEARED

enum **usbi_transfer_timeout_flags**

Enumerator

USBI_TRANSFER_OS_HANDLES_TIMEOUT

USBI_TRANSFER_TIMEOUT_HANDLED

USBI_TRANSFER_TIMED_OUT

34.233.3 Function Documentation

void usbi_log (struct **libusb_context** ∗ *ctx,* enum **libusb_log_level** *level,* **const** char ∗ *function,* **const** char ∗ *format,* ...)

void usbi_log_v (struct **libusb_context** ∗ *ctx,* enum **libusb_log_level** *level,* **const** char ∗ *function,* **const** char ∗ *format,* va_list *args*)

int usbi_io_init (struct **libusb_context** ∗ *ctx*)

void usbi_io_exit (struct **libusb_context** ∗ *ctx*)

struct **libusb_device**∗ usbi_alloc_device (struct **libusb_context** ∗ *ctx,* unsigned long *session_id*)

struct **libusb_device**∗ usbi_get_device_by_session_id (struct **libusb_context** ∗ *ctx,* unsigned long *session_id*)

int usbi_sanitize_device (struct **libusb_device** ∗ *dev*)

void usbi_handle_disconnect (struct **libusb_device_handle** ∗ *dev_handle*)

int usbi_handle_transfer_completion (struct **usbi_transfer** ∗ *itransfer,* enum **libusb_transfer_status** *status*)

int usbi_handle_transfer_cancellation (struct **usbi_transfer** ∗ *transfer*)

void usbi_signal_transfer_completion (struct **usbi_transfer** ∗ *transfer*)

int usbi_parse_descriptor (**const** unsigned char ∗ *source,* **const** char ∗ *descriptor,* **void** ∗ *dest,* int *host_endian*)

int usbi_device_cache_descriptor (**libusb_device** ∗ *dev*)

int usbi_get_config_index_by_value (struct **libusb_device** ∗ *dev,* **uint8_t** *bConfigurationValue,* int ∗ *idx*)

void usbi_connect_device (struct **libusb_device** ∗ *dev*)

void usbi_disconnect_device (struct **libusb_device** ∗ *dev*)

int usbi_signal_event (struct **libusb_context** ∗ *ctx*)

int usbi_clear_event (struct **libusb_context** ∗ *ctx*)

int usbi_add_pollfd (struct **libusb_context** ∗ *ctx,* int *fd,* short *events*)

void usbi_remove_pollfd (struct **libusb_context** ∗ *ctx,* int *fd*)

struct **discovered_devs**∗ discovered_devs_append (struct **discovered_devs** ∗ *discdevs,* struct **libusb_device** ∗ *dev*)

34.233.4 Variable Documentation

struct **libusb_context**∗ usbi_default_context

const struct **usbi_os_backend**∗ **const** usbi_backend

const struct **usbi_os_backend** linux_usbfs_backend

const struct **usbi_os_backend** darwin_backend

const struct **usbi_os_backend** openbsd_backend

const struct **usbi_os_backend** netbsd_backend

const struct **usbi_os_backend** windows_backend

const struct usbi_os_backend usbdk_backend

const struct usbi_os_backend wince_backend

const struct usbi_os_backend haiku_usb_raw_backend

const struct usbi_os_backend sunos_backend

struct list_head active_contexts_list

usbi_mutex_static_t active_contexts_lock

34.234 lib/ndkbuild/libusb/libusb/os/darwin_usb.c File Reference

```
#include "config.h"
#include <time.h>
#include <ctype.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <libkern/OSAtomic.h>
#include <sys/sysctl.h>
#include <mach/clock.h>
#include <mach/clock_types.h>
#include <mach/mach_host.h>
#include <mach/mach_port.h>
#include <AvailabilityMacros.h>
#include "darwin_usb.h"
```

Macros

- #define DARWIN_CACHED_DEVICE(a) ((struct darwin_cached_device ∗) (((struct darwin_device_priv ∗)((a)->os_priv))->dev))

Variables

- const struct usbi_os_backend darwin_backend

34.234.1 Macro Definition Documentation

#define DARWIN_CACHED_DEVICE(*a*) ((struct darwin_cached_device ∗) (((struct darwin_device_priv ∗)((a)->os_priv))->dev))

34.234.2 Variable Documentation

const struct usbi_os_backend darwin_backend

34.235 lib/ndkbuild/libusb/libusb/os/darwin_usb.h File Reference

```
#include "libusbi.h"
#include <IOKit/IOTypes.h>
#include <IOKit/IOCFBundle.h>
#include <IOKit/usb/IOUSBLib.h>
#include <IOKit/IOCFPlugIn.h>
```

Data Structures

- struct darwin_cached_device
- struct darwin_device_priv

- struct darwin_device_handle_priv
- struct darwin_device_handle_priv::darwin_interface
- struct darwin_transfer_priv

Macros

- #define IO_OBJECT_NULL ((io_object_t) 0)

Typedefs

- typedef IOCFPlugInInterface * io_cf_plugin_ref_t
- typedef IONotificationPortRef io_notification_port_t

34.235.1 Macro Definition Documentation

#define IO_OBJECT_NULL ((io_object_t) 0)

34.235.2 Typedef Documentation

typedef IOCFPlugInInterface* io_cf_plugin_ref_t

typedef IONotificationPortRef io_notification_port_t

34.236 lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp File Reference

```
#include "haiku_usb.h"
#include <cstdio>
#include <Directory.h>
#include <Entry.h>
#include <Looper.h>
#include <Messenger.h>
#include <Node.h>
#include <NodeMonitor.h>
#include <Path.h>
#include <cstring>
```

Data Structures

- class WatchedEntry
- class RosterLooper

34.237 lib/ndkbuild/libusb/libusb/os/haiku_usb.h File Reference

```
#include <List.h>
#include <Locker.h>
#include <Autolock.h>
#include <USBKit.h>
#include <map>
#include "libusbi.h"
#include "haiku_usb_raw.h"
```

Data Structures

- struct USBDevice
- class USBDeviceHandle
- class USBTransfer
- class USBRoster

34.238 lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp File Reference

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <new>
#include <vector>
#include "haiku_usb.h"
```

34.239 lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.cpp File Reference

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <new>
#include <vector>
#include "haiku_usb.h"
```

Variables

- USBRoster gUsbRoster
- int32 gInitCount = 0
- const struct usbi_os_backend haiku_usb_raw_backend

34.239.1 Variable Documentation

USBRoster gUsbRoster

int32 gInitCount = 0

const struct usbi_os_backend haiku_usb_raw_backend

34.240 lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.h File Reference

```
#include <USB3.h>
```

Data Structures

- union usb_raw_command

Macros

- #define B_USB_RAW_PROTOCOL_VERSION 0x0015
- #define B_USB_RAW_ACTIVE_ALTERNATE 0xffffffff

Enumerations

- enum usb_raw_command_id {
  B_USB_RAW_COMMAND_GET_VERSION = 0x1000, B_USB_RAW_COMMAND_GET_DEVICE_DESCRIPTOR = 0x2000, B_USB_RAW_COMMAND_GET_CONFIGURATION_DESCRIPTOR, B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR,
  B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR, B_USB_RAW_COMMAND_GET_STRING_DESCRIPTOR, B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR, B_USB_RAW_COMMAND_GET_ALT_INTERFACE_COUNT,
  B_USB_RAW_COMMAND_GET_ACTIVE_ALT_INTERFACE_INDEX, B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR_ETC, B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR_ETC, B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR_ETC,
  B_USB_RAW_COMMAND_SET_CONFIGURATION = 0x3000, B_USB_RAW_COMMAND_SET_FEATURE, B_USB_RAW_COMMAND_CLEAR_FEATURE, B_USB_RAW_COMMAND_GET_STATUS,
  B_USB_RAW_COMMAND_GET_DESCRIPTOR, B_USB_RAW_COMMAND_SET_ALT_INTERFACE, B_USB_RAW_COMMAND_CONTROL_TRANSFER = 0x4000, B_USB_RAW_COMMAND_INTERRUPT_TRANSFER,
  B_USB_RAW_COMMAND_BULK_TRANSFER, B_USB_RAW_COMMAND_ISOCHRONOUS_TRANSFER }

- enum usb_raw_command_status { B_USB_RAW_STATUS_SUCCESS = 0, B_USB_RAW_STATUS_FAILED, B_USB_RAW_STATUS_ABORTED, B_USB_RAW_STATUS_STALLED, B_USB_RAW_STATUS_CRC_ERROR, B_USB_RAW_STATUS_TIMEOUT, B_USB_RAW_STATUS_INVALID_CONFIGURATION, B_USB_RAW_STATUS_INVALID_INTERFACE, B_USB_RAW_STATUS_INVALID_ENDPOINT, B_USB_RAW_STATUS_INVALID_STRING, B_USB_RAW_STATUS_NO_MEMORY }

34.240.1 Macro Definition Documentation

#define B_USB_RAW_PROTOCOL_VERSION 0x0015

#define B_USB_RAW_ACTIVE_ALTERNATE 0xffffffff

34.240.2 Enumeration Type Documentation

enum **usb_raw_command_id**

Enumerator

B_USB_RAW_COMMAND_GET_VERSION

B_USB_RAW_COMMAND_GET_DEVICE_DESCRIPTOR

B_USB_RAW_COMMAND_GET_CONFIGURATION_DESCRIPTOR

B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR

B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR

B_USB_RAW_COMMAND_GET_STRING_DESCRIPTOR

B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR

B_USB_RAW_COMMAND_GET_ALT_INTERFACE_COUNT

B_USB_RAW_COMMAND_GET_ACTIVE_ALT_INTERFACE_INDEX

B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR_ETC

B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR_ETC

B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR_ETC

B_USB_RAW_COMMAND_SET_CONFIGURATION

B_USB_RAW_COMMAND_SET_FEATURE

B_USB_RAW_COMMAND_CLEAR_FEATURE

B_USB_RAW_COMMAND_GET_STATUS

B_USB_RAW_COMMAND_GET_DESCRIPTOR

B_USB_RAW_COMMAND_SET_ALT_INTERFACE

B_USB_RAW_COMMAND_CONTROL_TRANSFER

B_USB_RAW_COMMAND_INTERRUPT_TRANSFER

B_USB_RAW_COMMAND_BULK_TRANSFER

B_USB_RAW_COMMAND_ISOCHRONOUS_TRANSFER

enum **usb_raw_command_status**

Enumerator

B_USB_RAW_STATUS_SUCCESS

B_USB_RAW_STATUS_FAILED

B_USB_RAW_STATUS_ABORTED

B_USB_RAW_STATUS_STALLED

B_USB_RAW_STATUS_CRC_ERROR

B_USB_RAW_STATUS_TIMEOUT

B_USB_RAW_STATUS_INVALID_CONFIGURATION

B_USB_RAW_STATUS_INVALID_INTERFACE

B_USB_RAW_STATUS_INVALID_ENDPOINT

B_USB_RAW_STATUS_INVALID_STRING

B_USB_RAW_STATUS_NO_MEMORY

34.241 lib/ndkbuild/libusb/libusb/os/linux_netlink.c File Reference

```
#include <config.h>
#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <poll.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/netlink.h>
#include "libusbi.h"
#include "linux_usbfs.h"
```

Macros

- #define NL_GROUP_KERNEL 1

Functions

- int linux_netlink_start_event_monitor (void)
- int linux_netlink_stop_event_monitor (void)
- void linux_netlink_hotplug_poll (void)

34.241.1 Macro Definition Documentation

#define NL_GROUP_KERNEL 1

34.241.2 Function Documentation

int linux_netlink_start_event_monitor (**void**)

int linux_netlink_stop_event_monitor (**void**)

void linux_netlink_hotplug_poll (**void**)

34.242 lib/ndkbuild/libusb/libusb/os/linux_udev.c File Reference

```
#include <config.h>
#include <assert.h>
#include <ctype.h>
#include <dirent.h>
#include <errno.h>
#include <fcntl.h>
#include <poll.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/socket.h>
#include <unistd.h>
#include <libudev.h>
#include "libusbi.h"
#include "linux_usbfs.h"
```

Functions

- int linux_udev_start_event_monitor (void)
- int linux_udev_stop_event_monitor (void)

- int linux_udev_scan_devices (struct libusb_context *ctx)
- void linux_udev_hotplug_poll (void)

34.242.1 Function Documentation

int linux_udev_start_event_monitor (void)

int linux_udev_stop_event_monitor (void)

int linux_udev_scan_devices (struct libusb_context * *ctx*)

void linux_udev_hotplug_poll (void)

34.243 lib/ndkbuild/libusb/libusb/os/linux_usbfs.c File Reference

```
#include "config.h"
#include <assert.h>
#include <ctype.h>
#include <dirent.h>
#include <errno.h>
#include <fcntl.h>
#include <poll.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <time.h>
#include "libusbi.h"
#include "linux_usbfs.h"
```

Data Structures

- struct linux_device_priv
- struct linux_device_handle_priv
- struct linux_transfer_priv

Enumerations

- enum reap_action {
  NORMAL = 0, SUBMIT_FAILED, CANCELLED, COMPLETED_EARLY,
  ERROR }

Functions

- int linux_get_device_address (struct libusb_context *ctx, int detached, uint8_t *busnum, uint8_t *devaddr, const char *dev_node, const char *sys_name)
- int linux_enumerate_device (struct libusb_context *ctx, uint8_t busnum, uint8_t devaddr, const char *sysfs_dir)
- void linux_hotplug_enumerate (uint8_t busnum, uint8_t devaddr, const char *sys_name)
- void linux_device_disconnected (uint8_t busnum, uint8_t devaddr)

Variables

- usbi_mutex_static_t linux_hotplug_lock = USBI_MUTEX_INITIALIZER
- const struct usbi_os_backend linux_usbfs_backend

34.243.1 Enumeration Type Documentation

enum reap_action

Enumerator

NORMAL

SUBMIT_FAILED

CANCELLED

COMPLETED_EARLY

ERROR

34.243.2 Function Documentation

int linux_get_device_address (struct **libusb_context** ∗ *ctx,* int *detached,* **uint8_t** ∗ *busnum,* **uint8_t** ∗ *devaddr,* **const** char ∗ *dev_node,* **const** char ∗ *sys_name*)

int linux_enumerate_device (struct **libusb_context** ∗ *ctx,* **uint8_t** *busnum,* **uint8_t** *devaddr,* **const** char ∗ *sysfs_dir*)

void linux_hotplug_enumerate (**uint8_t** *busnum,* **uint8_t** *devaddr,* **const** char ∗ *sys_name*)

void linux_device_disconnected (**uint8_t** *busnum,* **uint8_t** *devaddr*)

34.243.3 Variable Documentation

usbi_mutex_static_t linux_hotplug_lock = **USBI_MUTEX_INITIALIZER**

const struct **usbi_os_backend** linux_usbfs_backend

34.244 lib/ndkbuild/libusb/libusb/os/linux_usbfs.h File Reference

```
#include <linux/types.h>
```

Data Structures

- struct usbfs_ctrltransfer
- struct usbfs_bulktransfer
- struct usbfs_setinterface
- struct usbfs_getdriver
- struct usbfs_iso_packet_desc
- struct usbfs_urb
- struct usbfs_connectinfo
- struct usbfs_ioctl
- struct usbfs_hub_portinfo
- struct usbfs_disconnect_claim
- struct usbfs_streams

Macros

- #define SYSFS_DEVICE_PATH "/sys/bus/usb/devices"
- #define USBFS_MAXDRIVERNAME 255
- #define USBFS_URB_SHORT_NOT_OK 0x01
- #define USBFS_URB_ISO_ASAP 0x02
- #define USBFS_URB_BULK_CONTINUATION 0x04
- #define USBFS_URB_QUEUE_BULK 0x10
- #define USBFS_URB_ZERO_PACKET 0x40
- #define MAX_ISO_BUFFER_LENGTH 49152 ∗ 128
- #define MAX_BULK_BUFFER_LENGTH 16384
- #define MAX_CTRL_BUFFER_LENGTH 4096
- #define USBFS_CAP_ZERO_PACKET 0x01
- #define USBFS_CAP_BULK_CONTINUATION 0x02
- #define USBFS_CAP_NO_PACKET_SIZE_LIM 0x04
- #define USBFS_CAP_BULK_SCATTER_GATHER 0x08
- #define USBFS_CAP_REAP_AFTER_DISCONNECT 0x10
- #define USBFS_DISCONNECT_CLAIM_IF_DRIVER 0x01
- #define USBFS_DISCONNECT_CLAIM_EXCEPT_DRIVER 0x02
- #define IOCTL_USBFS_CONTROL _IOWR('U', 0, struct usbfs_ctrltransfer)
- #define IOCTL_USBFS_BULK _IOWR('U', 2, struct usbfs_bulktransfer)
- #define IOCTL_USBFS_RESETEP _IOR('U', 3, unsigned int)
- #define IOCTL_USBFS_SETINTF _IOR('U', 4, struct usbfs_setinterface)
- #define IOCTL_USBFS_SETCONFIG _IOR('U', 5, unsigned int)

- #define IOCTL_USBFS_GETDRIVER _IOW('U', 8, struct usbfs_getdriver)
- #define IOCTL_USBFS_SUBMITURB _IOR('U', 10, struct usbfs_urb)
- #define IOCTL_USBFS_DISCARDURB _IO('U', 11)
- #define IOCTL_USBFS_REAPURB _IOW('U', 12, void *)
- #define IOCTL_USBFS_REAPURBNDELAY _IOW('U', 13, void *)
- #define IOCTL_USBFS_CLAIMINTF _IOR('U', 15, unsigned int)
- #define IOCTL_USBFS_RELEASEINTF _IOR('U', 16, unsigned int)
- #define IOCTL_USBFS_CONNECTINFO _IOW('U', 17, struct usbfs_connectinfo)
- #define IOCTL_USBFS_IOCTL _IOWR('U', 18, struct usbfs_ioctl)
- #define IOCTL_USBFS_HUB_PORTINFO _IOR('U', 19, struct usbfs_hub_portinfo)
- #define IOCTL_USBFS_RESET _IO('U', 20)
- #define IOCTL_USBFS_CLEAR_HALT _IOR('U', 21, unsigned int)
- #define IOCTL_USBFS_DISCONNECT _IO('U', 22)
- #define IOCTL_USBFS_CONNECT _IO('U', 23)
- #define IOCTL_USBFS_CLAIM_PORT _IOR('U', 24, unsigned int)
- #define IOCTL_USBFS_RELEASE_PORT _IOR('U', 25, unsigned int)
- #define IOCTL_USBFS_GET_CAPABILITIES _IOR('U', 26, __u32)
- #define IOCTL_USBFS_DISCONNECT_CLAIM _IOR('U', 27, struct usbfs_disconnect_claim)
- #define IOCTL_USBFS_ALLOC_STREAMS _IOR('U', 28, struct usbfs_streams)
- #define IOCTL_USBFS_FREE_STREAMS _IOR('U', 29, struct usbfs_streams)

Enumerations

- enum usbfs_urb_type { USBFS_URB_TYPE_ISO = 0, USBFS_URB_TYPE_INTERRUPT = 1, USBFS_URB_TYPE_CONTROL = 2, USBFS_URB_TYPE_BULK = 3 }

Functions

- int linux_netlink_start_event_monitor (void)
- int linux_netlink_stop_event_monitor (void)
- void linux_netlink_hotplug_poll (void)
- void linux_hotplug_enumerate (uint8_t busnum, uint8_t devaddr, const char *sys_name)
- void linux_device_disconnected (uint8_t busnum, uint8_t devaddr)
- int linux_get_device_address (struct libusb_context *ctx, int detached, uint8_t *busnum, uint8_t *devaddr, const char *dev_node, const char *sys_name)
- int linux_enumerate_device (struct libusb_context *ctx, uint8_t busnum, uint8_t devaddr, const char *sysfs_dir)

Variables

- usbi_mutex_static_t linux_hotplug_lock

34.244.1 Macro Definition Documentation

#define SYSFS_DEVICE_PATH "/sys/bus/usb/devices"

#define USBFS_MAXDRIVERNAME 255

#define USBFS_URB_SHORT_NOT_OK 0x01

#define USBFS_URB_ISO_ASAP 0x02

#define USBFS_URB_BULK_CONTINUATION 0x04

#define USBFS_URB_QUEUE_BULK 0x10

#define USBFS_URB_ZERO_PACKET 0x40

#define MAX_ISO_BUFFER_LENGTH 49152 * 128

#define MAX_BULK_BUFFER_LENGTH 16384

#define MAX_CTRL_BUFFER_LENGTH 4096

#define USBFS_CAP_ZERO_PACKET 0x01

#define USBFS_CAP_BULK_CONTINUATION 0x02

#define USBFS_CAP_NO_PACKET_SIZE_LIM 0x04

#define USBFS_CAP_BULK_SCATTER_GATHER 0x08

#define USBFS_CAP_REAP_AFTER_DISCONNECT 0x10

#define USBFS_DISCONNECT_CLAIM_IF_DRIVER 0x01

#define USBFS_DISCONNECT_CLAIM_EXCEPT_DRIVER 0x02

#define IOCTL_USBFS_CONTROL _IOWR('U', 0, struct usbfs_ctrltransfer)

#define IOCTL_USBFS_BULK _IOWR('U', 2, struct usbfs_bulktransfer)

#define IOCTL_USBFS_RESETEP _IOR('U', 3, unsigned int)

#define IOCTL_USBFS_SETINTF _IOR('U', 4, struct usbfs_setinterface)

#define IOCTL_USBFS_SETCONFIG _IOR('U', 5, unsigned int)

#define IOCTL_USBFS_GETDRIVER _IOW('U', 8, struct usbfs_getdriver)

#define IOCTL_USBFS_SUBMITURB _IOR('U', 10, struct usbfs_urb)

#define IOCTL_USBFS_DISCARDURB _IO('U', 11)

#define IOCTL_USBFS_REAPURB _IOW('U', 12, void *)

#define IOCTL_USBFS_REAPURBNDELAY _IOW('U', 13, void *)

#define IOCTL_USBFS_CLAIMINTF _IOR('U', 15, unsigned int)

#define IOCTL_USBFS_RELEASEINTF _IOR('U', 16, unsigned int)

#define IOCTL_USBFS_CONNECTINFO _IOW('U', 17, struct usbfs_connectinfo)

#define IOCTL_USBFS_IOCTL _IOWR('U', 18, struct usbfs_ioctl)

#define IOCTL_USBFS_HUB_PORTINFO _IOR('U', 19, struct usbfs_hub_portinfo)

#define IOCTL_USBFS_RESET _IO('U', 20)

#define IOCTL_USBFS_CLEAR_HALT _IOR('U', 21, unsigned int)

#define IOCTL_USBFS_DISCONNECT _IO('U', 22)

#define IOCTL_USBFS_CONNECT _IO('U', 23)

#define IOCTL_USBFS_CLAIM_PORT _IOR('U', 24, unsigned int)

#define IOCTL_USBFS_RELEASE_PORT _IOR('U', 25, unsigned int)

#define IOCTL_USBFS_GET_CAPABILITIES _IOR('U', 26, __u32)

#define IOCTL_USBFS_DISCONNECT_CLAIM _IOR('U', 27, struct usbfs_disconnect_claim)

#define IOCTL_USBFS_ALLOC_STREAMS _IOR('U', 28, struct usbfs_streams)

#define IOCTL_USBFS_FREE_STREAMS _IOR('U', 29, struct usbfs_streams)

34.244.2 Enumeration Type Documentation

enum usbfs_urb_type

Enumerator

USBFS_URB_TYPE_ISO

USBFS_URB_TYPE_INTERRUPT

USBFS_URB_TYPE_CONTROL

USBFS_URB_TYPE_BULK

34.244.3 Function Documentation

int linux_netlink_start_event_monitor (void)

int linux_netlink_stop_event_monitor (void)

void linux_netlink_hotplug_poll (void)

void linux_hotplug_enumerate (uint8_t *busnum,* **uint8_t** *devaddr,* **const** char ∗ *sys_name*)

void linux_device_disconnected (uint8_t *busnum,* **uint8_t** *devaddr*)

int linux_get_device_address (struct libusb_context ∗ *ctx,* **int** *detached,* **uint8_t** ∗ *busnum,* **uint8_t** ∗ *devaddr,* **const** char ∗ *dev_node,* **const** char ∗ *sys_name*)

int linux_enumerate_device (struct libusb_context ∗ *ctx,* **uint8_t** *busnum,* **uint8_t** *devaddr,* **const** char ∗ *sysfs_dir*)

34.244.4 Variable Documentation

usbi_mutex_static_t linux_hotplug_lock

34.245 lib/ndkbuild/libusb/libusb/os/netbsd_usb.c File Reference

```
#include <config.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dev/usb/usb.h>
#include "libusbi.h"
```

Data Structures

- struct device_priv
- struct handle_priv

Variables

- const struct usbi_os_backend netbsd_backend

34.245.1 Variable Documentation

const struct usbi_os_backend netbsd_backend

34.246 lib/ndkbuild/libusb/libusb/os/openbsd_usb.c File Reference

```
#include <config.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dev/usb/usb.h>
#include "libusbi.h"
```

Data Structures

- struct device_priv
- struct handle_priv

Macros

- #define DEVPATH "/dev/"
- #define USBDEV DEVPATH "usb"

Variables

- const struct usbi_os_backend openbsd_backend

34.246.1 Macro Definition Documentation

#define DEVPATH "/dev/"

#define USBDEV DEVPATH "usb"

34.246.2 Variable Documentation

const struct usbi_os_backend openbsd_backend

34.247 lib/ndkbuild/libusb/libusb/os/poll_posix.c File Reference

```
#include <config.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include "libusbi.h"
```

Functions

- int usbi_pipe (int pipefd[2])

34.247.1 Function Documentation

int usbi_pipe (int *pipefd[2]*)

34.248 lib/ndkbuild/libusb/libusb/os/poll_posix.h File Reference

Macros

- #define usbi_write write
- #define usbi_read read
- #define usbi_close close
- #define usbi_poll poll

Functions

- int usbi_pipe (int pipefd[2])

34.248.1 Macro Definition Documentation

#define usbi_write write

#define usbi_read read

#define usbi_close close

#define usbi_poll poll

34.248.2 Function Documentation

int usbi_pipe (int *pipefd[2]*)

34.249 lib/ndkbuild/libusb/libusb/os/poll_windows.c File Reference

```
#include <config.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include "libusbi.h"
```

Macros

- #define poll_dbg(...)
- #define CHECK_INIT_POLLING do {if(!is_polling_set) init_polling();} while(0)
- #define Use_Duplicate_Handles (pCancelIoEx == NULL)

Functions

- void init_polling (void)
- void exit_polling (void)
- int usbi_pipe (int filedes[2])
- struct winfd usbi_create_fd (HANDLE handle, int access_mode, struct usbi_transfer *itransfer, cancel_transfer *cancel_fn)
- void usbi_free_fd (struct winfd *wfd)
- struct winfd fd_to_winfd (int fd)
- struct winfd handle_to_winfd (HANDLE handle)
- struct winfd overlapped_to_winfd (OVERLAPPED *overlapped)
- int usbi_poll (struct pollfd *fds, unsigned int nfds, int timeout)
- int usbi_close (int fd)
- ssize_t usbi_write (int fd, const void *buf, size_t count)
- ssize_t usbi_read (int fd, void *buf, size_t count)

Variables

- const struct winfd INVALID_WINFD = {-1, INVALID_HANDLE_VALUE, NULL, NULL, NULL, RW_NONE}
- struct winfd poll_fd [MAX_FDS]
- BOOLEAN is_polling_set = FALSE
- LONG pipe_number = 0
- static LPOVERLAPPED = NULL

34.249.1 Macro Definition Documentation

#define poll_dbg(...)

#define CHECK_INIT_POLLING do {if(!is_polling_set) init_polling();} while(0)

#define Use_Duplicate_Handles (pCancelIoEx == NULL)

34.249.2 Function Documentation

void init_polling (void)

void exit_polling (void)

int usbi_pipe (int *filedes[2]*)

struct winfd usbi_create_fd (HANDLE *handle,* int *access_mode,* struct usbi_transfer * *itransfer,* cancel_transfer * *cancel_fn*)

void usbi_free_fd (struct winfd * *wfd*)

struct winfd fd_to_winfd (int *fd*)

struct winfd handle_to_winfd (HANDLE *handle*)

struct winfd overlapped_to_winfd (OVERLAPPED * *overlapped*)

int usbi_poll (struct pollfd * *fds,* unsigned int *nfds,* int *timeout*)

int usbi_close (int *fd*)

ssize_t usbi_write (int *fd,* const void * *buf,* size_t *count*)

ssize_t usbi_read (int *fd,* void * *buf,* size_t *count*)

34.249.3 Variable Documentation

const struct winfd INVALID_WINFD = {-1, INVALID_HANDLE_VALUE, NULL, NULL, NULL, RW_NONE}

struct winfd poll_fd[MAX_FDS]

CRITICAL_SECTION mutex

HANDLE original_handle

DWORD thread_id

BOOLEAN is_polling_set = FALSE

LONG pipe_number = 0

LPOVERLAPPED = NULL

34.250 lib/ndkbuild/libusb/libusb/os/poll_windows.h File Reference

Data Structures

- struct pollfd
- struct winfd

Macros

- #define STATUS_REPARSE ((LONG)0x00000104L)
- #define STATUS_COMPLETED_SYNCHRONOUSLY STATUS_REPARSE
- #define HasOverlappedIoCompletedSync(lpOverlapped) (((DWORD)(lpOverlapped)->Internal) == STATUS_COMPLETED_SYNCHRONOUSLY)
- #define DUMMY_HANDLE ((HANDLE)(LONG_PTR)-2)
- #define MAX_FDS 256
- #define POLLIN 0x0001 /* There is data to read */
- #define POLLPRI 0x0002 /* There is urgent data to read */
- #define POLLOUT 0x0004 /* Writing now will not block */
- #define POLLERR 0x0008 /* Error condition */
- #define POLLHUP 0x0010 /* Hung up */
- #define POLLNVAL 0x0020 /* Invalid request: fd not open */
- #define TIMESPEC_TO_TIMEVAL(tv, ts)
- #define timersub(a, b, result)

Typedefs

- typedef int cancel_transfer (struct usbi_transfer *itransfer)

Enumerations

- enum windows_version {
  WINDOWS_CE = -2, WINDOWS_UNDEFINED = -1, WINDOWS_UNSUPPORTED = 0, WINDOWS_XP = 0x51, WINDOWS_2003 = 0x52, WINDOWS_VISTA = 0x60, WINDOWS_7 = 0x61, WINDOWS_8 = 0x62, WINDOWS_8_1_OR_LATER = 0x63, WINDOWS_MAX }
- enum rw_type { RW_NONE, RW_READ, RW_WRITE }

Functions

- int usbi_pipe (int pipefd[2])
- int usbi_poll (struct pollfd ∗fds, unsigned int nfds, int timeout)
- ssize_t usbi_write (int fd, const void ∗buf, size_t count)
- ssize_t usbi_read (int fd, void ∗buf, size_t count)
- int usbi_close (int fd)
- void init_polling (void)
- void exit_polling (void)
- struct winfd usbi_create_fd (HANDLE handle, int access_mode, struct usbi_transfer ∗transfer, cancel_transfer ∗cancel_fn)
- void usbi_free_fd (struct winfd ∗winfd)
- struct winfd fd_to_winfd (int fd)
- struct winfd handle_to_winfd (HANDLE handle)
- struct winfd overlapped_to_winfd (OVERLAPPED ∗overlapped)

Variables

- int windows_version
- const struct winfd INVALID_WINFD

34.250.1 Macro Definition Documentation

#define STATUS_REPARSE ((LONG)0x00000104L)

#define STATUS_COMPLETED_SYNCHRONOUSLY STATUS_REPARSE

#define HasOverlappedIoCompletedSync(*lpOverlapped*) (((DWORD)(lpOverlapped)->Internal) == STATUS_COMPLETED_SYNCHRONOUSLY)

#define DUMMY_HANDLE ((HANDLE)(LONG_PTR)-2)

#define MAX_FDS 256

#define POLLIN 0x0001 /∗ There is data to read ∗/

#define POLLPRI 0x0002 /∗ There is urgent data to read ∗/

#define POLLOUT 0x0004 /∗ Writing now will not block ∗/

#define POLLERR 0x0008 /∗ Error condition ∗/

#define POLLHUP 0x0010 /∗ Hung up ∗/

#define POLLNVAL 0x0020 /∗ Invalid request: fd not open ∗/

#define TIMESPEC_TO_TIMEVAL(*tv, ts*)

Value:

```
{                   \
        (tv)->tv_sec = (long)(ts)->tv_sec;                  \
        (tv)->tv_usec = (long)(ts)->tv_nsec / 1000;         \
}
```

#define timersub(*a, b, result*)

Value:

```
do {                                                        \
        (result)->tv_sec = (a)->tv_sec - (b)->tv_sec;       \
        (result)->tv_usec = (a)->tv_usec - (b)->tv_usec;    \
        if ((result)->tv_usec < 0) {                        \
                --(result)->tv_sec;                             \
                (result)->tv_usec += 1000000;                   \
        }                                                   \
} while (0)
```

34.250.2 Typedef Documentation

typedef int cancel_transfer(struct **usbi_transfer** ∗itransfer)

34.250.3 Enumeration Type Documentation

enum **windows_version**

Enumerator

- ***WINDOWS_CE***
- ***WINDOWS_UNDEFINED***
- ***WINDOWS_UNSUPPORTED***
- ***WINDOWS_XP***
- ***WINDOWS_2003***
- ***WINDOWS_VISTA***
- ***WINDOWS_7***
- ***WINDOWS_8***
- ***WINDOWS_8_1_OR_LATER***
- ***WINDOWS_MAX***

enum **rw_type**

Enumerator

- ***RW_NONE***
- ***RW_READ***
- ***RW_WRITE***

34.250.4 Function Documentation

int usbi_pipe (int *pipefd[2]*)

int usbi_poll (struct **pollfd** ∗ *fds,* unsigned int *nfds,* int *timeout*)

ssize_t usbi_write (int *fd,* **const void** ∗ *buf,* size_t *count*)

ssize_t usbi_read (int *fd,* **void** ∗ *buf,* size_t *count*)

int usbi_close (int *fd*)

void init_polling (**void**)

void exit_polling (**void**)

struct **winfd** usbi_create_fd (HANDLE *handle,* int *access_mode,* struct **usbi_transfer** ∗ *transfer,* **cancel_transfer** ∗ *cancel_fn*)

void usbi_free_fd (struct **winfd** ∗ *winfd*)

struct **winfd** fd_to_winfd (int *fd*)

struct **winfd** handle_to_winfd (HANDLE *handle*)

struct **winfd** overlapped_to_winfd (OVERLAPPED ∗ *overlapped*)

34.250.5 Variable Documentation

int **windows_version**

const struct **winfd** INVALID_WINFD

34.251 lib/ndkbuild/libusb/libusb/os/sunos_usb.c File Reference

```
#include <config.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <strings.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <aio.h>
#include <libdevinfo.h>
#include <sys/usb/clients/ugen/usb_ugen.h>
#include <sys/usb/usba.h>
#include <sys/pci.h>
#include "libusbi.h"
#include "sunos_usb.h"
```

Variables

- const struct usbi_os_backend sunos_backend

34.251.1 Variable Documentation

const struct usbi_os_backend sunos_backend

34.252 lib/ndkbuild/libusb/libusb/os/sunos_usb.h File Reference

```
#include <libdevinfo.h>
#include <pthread.h>
#include "libusbi.h"
```

Data Structures

- struct sunos_device_priv
- struct endpoint
- struct sunos_device_handle_priv
- struct sunos_transfer_priv
- struct node_args
- struct devlink_cbarg
- struct aio_callback_args

Macros

- #define READ 0
- #define WRITE 1

Typedefs

- typedef struct sunos_device_priv sunos_dev_priv_t
- typedef struct endpoint sunos_ep_priv_t
- typedef struct sunos_device_handle_priv sunos_dev_handle_priv_t
- typedef struct sunos_transfer_priv sunos_xfer_priv_t

34.252.1 Macro Definition Documentation

#define READ 0

#define WRITE 1

34.252.2 Typedef Documentation

typedef struct sunos_device_priv sunos_dev_priv_t

typedef struct endpoint sunos_ep_priv_t

typedef struct sunos_device_handle_priv sunos_dev_handle_priv_t

typedef struct sunos_transfer_priv sunos_xfer_priv_t

34.253 lib/ndkbuild/libusb/libusb/os/threads_posix.c File Reference

```
#include <config.h>
#include <time.h>
#include "threads_posix.h"
#include "libusbi.h"
```

Functions

- int usbi_cond_timedwait (pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timeval *tv)
- int usbi_get_tid (void)

34.253.1 Function Documentation

int usbi_cond_timedwait (pthread_cond_t * *cond,* pthread_mutex_t * *mutex,* const struct timeval * *tv*)

int usbi_get_tid (void)

34.254 lib/ndkbuild/libusb/libusb/os/threads_posix.h File Reference

```
#include <pthread.h>
```

Macros

- #define usbi_mutex_static_t pthread_mutex_t
- #define USBI_MUTEX_INITIALIZER PTHREAD_MUTEX_INITIALIZER
- #define usbi_mutex_static_lock pthread_mutex_lock
- #define usbi_mutex_static_unlock pthread_mutex_unlock
- #define usbi_mutex_t pthread_mutex_t
- #define usbi_mutex_init(mutex) pthread_mutex_init((mutex), NULL)
- #define usbi_mutex_lock pthread_mutex_lock
- #define usbi_mutex_unlock pthread_mutex_unlock
- #define usbi_mutex_trylock pthread_mutex_trylock
- #define usbi_mutex_destroy pthread_mutex_destroy
- #define usbi_cond_t pthread_cond_t
- #define usbi_cond_init(cond) pthread_cond_init((cond), NULL)
- #define usbi_cond_wait pthread_cond_wait
- #define usbi_cond_broadcast pthread_cond_broadcast
- #define usbi_cond_destroy pthread_cond_destroy
- #define usbi_tls_key_t pthread_key_t
- #define usbi_tls_key_create(key) pthread_key_create((key), NULL)
- #define usbi_tls_key_get pthread_getspecific
- #define usbi_tls_key_set pthread_setspecific
- #define usbi_tls_key_delete pthread_key_delete

Functions

- int usbi_cond_timedwait (pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timeval *tv)
- int usbi_get_tid (void)

34.254.1 Macro Definition Documentation

#define usbi_mutex_static_t pthread_mutex_t

#define USBI_MUTEX_INITIALIZER PTHREAD_MUTEX_INITIALIZER

#define usbi_mutex_static_lock pthread_mutex_lock

#define usbi_mutex_static_unlock pthread_mutex_unlock

#define usbi_mutex_t pthread_mutex_t

#define usbi_mutex_init(mutex) pthread_mutex_init((mutex), NULL)

#define usbi_mutex_lock pthread_mutex_lock

#define usbi_mutex_unlock pthread_mutex_unlock

#define usbi_mutex_trylock pthread_mutex_trylock

#define usbi_mutex_destroy pthread_mutex_destroy

#define usbi_cond_t pthread_cond_t

#define usbi_cond_init(*cond*) pthread_cond_init((cond), NULL)

#define usbi_cond_wait pthread_cond_wait

#define usbi_cond_broadcast pthread_cond_broadcast

#define usbi_cond_destroy pthread_cond_destroy

#define usbi_tls_key_t pthread_key_t

#define usbi_tls_key_create(*key*) pthread_key_create((key), NULL)

#define usbi_tls_key_get pthread_getspecific

#define usbi_tls_key_set pthread_setspecific

#define usbi_tls_key_delete pthread_key_delete

34.254.2 Function Documentation

int usbi_cond_timedwait (pthread_cond_t * *cond,* pthread_mutex_t * *mutex,* const struct timeval * *tv*)

int usbi_get_tid (void)

34.255 lib/ndkbuild/libusb/libusb/os/threads_windows.c File Reference

```
#include <config.h>
#include <objbase.h>
#include <errno.h>
#include "libusbi.h"
```

Data Structures

- struct usbi_cond_perthread

Functions

- int usbi_mutex_static_lock (usbi_mutex_static_t *mutex)
- int usbi_mutex_static_unlock (usbi_mutex_static_t *mutex)
- int usbi_mutex_init (usbi_mutex_t *mutex)
- int usbi_mutex_lock (usbi_mutex_t *mutex)
- int usbi_mutex_unlock (usbi_mutex_t *mutex)
- int usbi_mutex_trylock (usbi_mutex_t *mutex)
- int usbi_mutex_destroy (usbi_mutex_t *mutex)
- int usbi_cond_init (usbi_cond_t *cond)
- int usbi_cond_destroy (usbi_cond_t *cond)
- int usbi_cond_broadcast (usbi_cond_t *cond)
- int usbi_cond_wait (usbi_cond_t *cond, usbi_mutex_t *mutex)
- int usbi_cond_timedwait (usbi_cond_t *cond, usbi_mutex_t *mutex, const struct timeval *tv)
- int usbi_tls_key_create (usbi_tls_key_t *key)
- void * usbi_tls_key_get (usbi_tls_key_t key)
- int usbi_tls_key_set (usbi_tls_key_t key, void *value)
- int usbi_tls_key_delete (usbi_tls_key_t key)
- int usbi_get_tid (void)

34.255.1 Function Documentation

int usbi_mutex_static_lock (**usbi_mutex_static_t** * *mutex*)

int usbi_mutex_static_unlock (**usbi_mutex_static_t** * *mutex*)

int usbi_mutex_init (**usbi_mutex_t** * *mutex*)

int usbi_mutex_lock (**usbi_mutex_t** * *mutex*)

int usbi_mutex_unlock (**usbi_mutex_t** * *mutex*)

int usbi_mutex_trylock (**usbi_mutex_t** * *mutex*)

int usbi_mutex_destroy (**usbi_mutex_t** * *mutex*)

int usbi_cond_init (**usbi_cond_t** * *cond*)

int usbi_cond_destroy (**usbi_cond_t** * *cond*)

int usbi_cond_broadcast (**usbi_cond_t** * *cond*)

int usbi_cond_wait (**usbi_cond_t** * *cond,* **usbi_mutex_t** * *mutex*)

int usbi_cond_timedwait (**usbi_cond_t** * *cond,* **usbi_mutex_t** * *mutex,* **const** struct timeval * *tv*)

int usbi_tls_key_create (**usbi_tls_key_t** * *key*)

void* usbi_tls_key_get (**usbi_tls_key_t** *key*)

int usbi_tls_key_set (**usbi_tls_key_t** *key,* **void** * *value*)

int usbi_tls_key_delete (**usbi_tls_key_t** *key*)

int usbi_get_tid (**void**)

34.256 lib/ndkbuild/libusb/libusb/os/threads_windows.h File Reference

Data Structures

- struct usbi_cond
- struct timespec

Macros

- #define usbi_mutex_static_t volatile LONG
- #define USBI_MUTEX_INITIALIZER 0
- #define usbi_mutex_t HANDLE
- #define HAVE_STRUCT_TIMESPEC 1
- #define ETIMEDOUT 10060 /∗ This is the value in winsock.h. ∗/
- #define usbi_tls_key_t DWORD

Typedefs

- typedef struct usbi_cond usbi_cond_t

Functions

- int usbi_mutex_static_lock (usbi_mutex_static_t ∗mutex)
- int usbi_mutex_static_unlock (usbi_mutex_static_t ∗mutex)
- int usbi_mutex_init (usbi_mutex_t ∗mutex)
- int usbi_mutex_lock (usbi_mutex_t ∗mutex)
- int usbi_mutex_unlock (usbi_mutex_t ∗mutex)
- int usbi_mutex_trylock (usbi_mutex_t ∗mutex)
- int usbi_mutex_destroy (usbi_mutex_t ∗mutex)
- int usbi_cond_init (usbi_cond_t ∗cond)
- int usbi_cond_wait (usbi_cond_t ∗cond, usbi_mutex_t ∗mutex)
- int usbi_cond_timedwait (usbi_cond_t ∗cond, usbi_mutex_t ∗mutex, const struct timeval ∗tv)
- int usbi_cond_broadcast (usbi_cond_t ∗cond)
- int usbi_cond_destroy (usbi_cond_t ∗cond)
- int usbi_tls_key_create (usbi_tls_key_t ∗key)
- void ∗ usbi_tls_key_get (usbi_tls_key_t key)
- int usbi_tls_key_set (usbi_tls_key_t key, void ∗value)
- int usbi_tls_key_delete (usbi_tls_key_t key)
- int usbi_get_tid (void)

34.256.1 Macro Definition Documentation

#define usbi_mutex_static_t volatile LONG

#define USBI_MUTEX_INITIALIZER 0

#define usbi_mutex_t HANDLE

#define HAVE_STRUCT_TIMESPEC 1

#define ETIMEDOUT 10060 /∗ This is the value in winsock.h. ∗/

#define usbi_tls_key_t DWORD

34.256.2 Typedef Documentation

typedef struct usbi_cond usbi_cond_t

34.256.3 Function Documentation

int usbi_mutex_static_lock (usbi_mutex_static_t ∗ *mutex*)

int usbi_mutex_static_unlock (usbi_mutex_static_t ∗ *mutex*)

int usbi_mutex_init (usbi_mutex_t ∗ *mutex*)

int usbi_mutex_lock (usbi_mutex_t ∗ *mutex*)

int usbi_mutex_unlock (usbi_mutex_t ∗ *mutex*)

int usbi_mutex_trylock (usbi_mutex_t ∗ *mutex*)

int usbi_mutex_destroy (usbi_mutex_t ∗ *mutex*)

int usbi_cond_init (usbi_cond_t * *cond*)

int usbi_cond_wait (usbi_cond_t * *cond,* usbi_mutex_t * *mutex*)

int usbi_cond_timedwait (usbi_cond_t * *cond,* usbi_mutex_t * *mutex,* const struct timeval * *tv*)

int usbi_cond_broadcast (usbi_cond_t * *cond*)

int usbi_cond_destroy (usbi_cond_t * *cond*)

int usbi_tls_key_create (usbi_tls_key_t * *key*)

void* usbi_tls_key_get (usbi_tls_key_t *key*)

int usbi_tls_key_set (usbi_tls_key_t *key,* void * *value*)

int usbi_tls_key_delete (usbi_tls_key_t *key*)

int usbi_get_tid (void)

34.257 lib/ndkbuild/libusb/libusb/os/wince_usb.c File Reference

```
#include <config.h>
#include <stdint.h>
#include <inttypes.h>
#include "libusbi.h"
#include "wince_usb.h"
```

Variables

- const uint64_t epoch_time = UINT64_C(116444736000000000)
- int windows_version = WINDOWS_CE
- const struct usbi_os_backend wince_backend

34.257.1 Variable Documentation

const uint64_t epoch_time = UINT64_C(116444736000000000)

int windows_version = WINDOWS_CE

const struct usbi_os_backend wince_backend

34.258 lib/ndkbuild/libusb/libusb/os/wince_usb.h File Reference

```
#include "windows_common.h"
#include <windows.h>
#include "poll_windows.h"
```

Data Structures

- struct UKW_DEVICE_DESCRIPTOR
- struct UKW_CONTROL_HEADER
- struct wince_device_priv
- struct wince_transfer_priv

Macros

- #define MAX_DEVICE_COUNT 256
- #define UKW_TF_IN_TRANSFER 0x00000001
- #define UKW_TF_OUT_TRANSFER 0x00000000
- #define UKW_TF_NO_WAIT 0x00000100
- #define UKW_TF_SHORT_TRANSFER_OK 0x00000200
- #define UKW_TF_SEND_TO_DEVICE 0x00010000

- #define UKW_TF_SEND_TO_INTERFACE 0x00020000
- #define UKW_TF_SEND_TO_ENDPOINT 0x00040000
- #define UKW_TF_DONT_BLOCK_FOR_MEM 0x00080000
- #define UKW_ACTIVE_CONFIGURATION -1
- #define STATUS_HALT_FLAG 0x1

Typedefs

- typedef struct UKW_DEVICE_PRIV * UKW_DEVICE
- typedef UKW_DEVICE * PUKW_DEVICE
- typedef UKW_DEVICE * LPUKW_DEVICE
- typedef struct UKW_DEVICE_DESCRIPTOR * PUKW_DEVICE_DESCRIPTOR
- typedef struct UKW_DEVICE_DESCRIPTOR * LPUKW_DEVICE_DESCRIPTOR
- typedef struct UKW_CONTROL_HEADER * PUKW_CONTROL_HEADER
- typedef struct UKW_CONTROL_HEADER * LPUKW_CONTROL_HEADER

Functions

- DLL_DECLARE_HANDLE (ceusbkwrapper)
- DLL_DECLARE_FUNC (WINAPI, HANDLE, UkwOpenDriver,())
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwGetDeviceList,(HANDLE, LPUKW_DEVICE, DWORD, LPDWORD))
- DLL_DECLARE_FUNC (WINAPI, void, UkwReleaseDeviceList,(HANDLE, LPUKW_DEVICE, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwGetDeviceAddress,(UKW_DEVICE, unsigned char *, unsigned char *, unsigned long *))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwGetDeviceDescriptor,(UKW_DEVICE, LPUKW_DEVICE_DESCRIPTOR))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwGetConfigDescriptor,(UKW_DEVICE, DWORD, LPVOID, DWORD, LPDWORD))
- DLL_DECLARE_FUNC (WINAPI, void, UkwCloseDriver,(HANDLE))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwCancelTransfer,(UKW_DEVICE, LPOVERLAPPED, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwIssueControlTransfer,(UKW_DEVICE, DWORD, LPUKW_CONTROL_HEADER, LPVOID, DWORD, LPDWORD, LPOVERLAPPED))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwClaimInterface,(UKW_DEVICE, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwReleaseInterface,(UKW_DEVICE, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwSetInterfaceAlternateSetting,(UKW_DEVICE, DWORD, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwClearHaltHost,(UKW_DEVICE, UCHAR))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwClearHaltDevice,(UKW_DEVICE, UCHAR))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwGetConfig,(UKW_DEVICE, PUCHAR))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwSetConfig,(UKW_DEVICE, UCHAR))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwResetDevice,(UKW_DEVICE))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwKernelDriverActive,(UKW_DEVICE, DWORD, PBOOL))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwAttachKernelDriver,(UKW_DEVICE, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwDetachKernelDriver,(UKW_DEVICE, DWORD))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwIssueBulkTransfer,(UKW_DEVICE, DWORD, UCHAR, LPVOID, DWORD, LPDWORD, LPOVERLAPPED))
- DLL_DECLARE_FUNC (WINAPI, BOOL, UkwIsPipeHalted,(UKW_DEVICE, UCHAR, LPBOOL))

34.258.1 Macro Definition Documentation

#define MAX_DEVICE_COUNT 256

#define UKW_TF_IN_TRANSFER 0x00000001

#define UKW_TF_OUT_TRANSFER 0x00000000

#define UKW_TF_NO_WAIT 0x00000100

#define UKW_TF_SHORT_TRANSFER_OK 0x00000200

#define UKW_TF_SEND_TO_DEVICE 0x00010000

#define UKW_TF_SEND_TO_INTERFACE 0x00020000

#define UKW_TF_SEND_TO_ENDPOINT 0x00040000

#define UKW_TF_DONT_BLOCK_FOR_MEM 0x00080000

#define UKW_ACTIVE_CONFIGURATION -1

#define STATUS_HALT_FLAG 0x1

34.258.2 Typedef Documentation

typedef struct UKW_DEVICE_PRIV* **UKW_DEVICE**

typedef **UKW_DEVICE*** **PUKW_DEVICE**

typedef **UKW_DEVICE** * **LPUKW_DEVICE**

typedef struct **UKW_DEVICE_DESCRIPTOR** * **PUKW_DEVICE_DESCRIPTOR**

typedef struct **UKW_DEVICE_DESCRIPTOR** * **LPUKW_DEVICE_DESCRIPTOR**

typedef struct **UKW_CONTROL_HEADER** * **PUKW_CONTROL_HEADER**

typedef struct **UKW_CONTROL_HEADER** * **LPUKW_CONTROL_HEADER**

34.258.3 Function Documentation

DLL_DECLARE_HANDLE (ceusbkwrapper)

DLL_DECLARE_FUNC (WINAPI , HANDLE , UkwOpenDriver , ())

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwGetDeviceList , (HANDLE, **LPUKW_DEVICE**, DWORD, LPDWORD))

DLL_DECLARE_FUNC (WINAPI , **void** , UkwReleaseDeviceList , (HANDLE, **LPUKW_DEVICE**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwGetDeviceAddress , (**UKW_DEVICE**, unsigned char *, unsigned char *, unsigned long *))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwGetDeviceDescriptor , (**UKW_DEVICE**, **LPUKW_DEVICE_DESCRIPTOR**))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwGetConfigDescriptor , (**UKW_DEVICE**, DWORD, LPVOID, DWORD, LPDWORD))

DLL_DECLARE_FUNC (WINAPI , **void** , UkwCloseDriver , (HANDLE))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwCancelTransfer , (**UKW_DEVICE**, **LPOVERLAPPED**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwIssueControlTransfer , (**UKW_DEVICE**, DWORD, **LPUKW_CONTROL_HEADER**, LPVOID, DWORD, LPDWORD, **LPOVERLAPPED**))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwClaimInterface , (**UKW_DEVICE**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwReleaseInterface , (**UKW_DEVICE**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwSetInterfaceAlternateSetting , (**UKW_DEVICE**, DWORD, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwClearHaltHost , (**UKW_DEVICE**, UCHAR))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwClearHaltDevice , (**UKW_DEVICE**, UCHAR))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwGetConfig , (**UKW_DEVICE**, PUCHAR))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwSetConfig , (**UKW_DEVICE**, UCHAR))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwResetDevice , (**UKW_DEVICE**))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwKernelDriverActive , (**UKW_DEVICE**, DWORD, PBOOL))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwAttachKernelDriver , (**UKW_DEVICE**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwDetachKernelDriver , (**UKW_DEVICE**, DWORD))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwIssueBulkTransfer , (UKW_DEVICE, DWORD, UCHAR, LPVOID, DWORD, LPDWORD, LPOVERLAPPED))

DLL_DECLARE_FUNC (WINAPI , BOOL , UkwIsPipeHalted , (UKW_DEVICE, UCHAR, LPBOOL))

34.259 lib/ndkbuild/libusb/libusb/os/windows_common.h File Reference

Macros

- #define bool BOOL
- #define true TRUE
- #define false FALSE
- #define safe_free(p) do {if (p != NULL) {free((void*)p); p = NULL;}} while(0)
- #define safe_closehandle(h) do {if (h != INVALID_HANDLE_VALUE) {CloseHandle(h); h = INVALID_HANDLE_VALUE;}} while(0)
- #define safe_min(a, b) MIN((size_t)(a), (size_t)(b))
- #define safe_strcp(dst, dst_max, src, count)
- #define safe_strcpy(dst, dst_max, src) safe_strcp(dst, dst_max, src, safe_strlen(src)+1)
- #define safe_strncat(dst, dst_max, src, count) strncat(dst, src, safe_min(count, dst_max - safe_strlen(dst) - 1))
- #define safe_strcat(dst, dst_max, src) safe_strncat(dst, dst_max, src, safe_strlen(src)+1)
- #define safe_strcmp(str1, str2) strcmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2))
- #define safe_stricmp(str1, str2) _stricmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2))
- #define safe_strncmp(str1, str2, count) strncmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2), count)
- #define safe_strlen(str) ((str==NULL)?0:strlen(str))
- #define safe_sprintf(dst, count,...) do {_snprintf(dst, count, __VA_ARGS__); (dst)[(count)-1] = 0; } while(0)
- #define safe_stprintf _sntprintf
- #define safe_tcslen(str) ((str==NULL)?0:_tcslen(str))
- #define safe_unref_device(dev) do {if (dev != NULL) {libusb_unref_device(dev); dev = NULL;}} while(0)
- #define wchar_to_utf8_ms(wstr, str, strlen) WideCharToMultiByte(CP_UTF8, 0, wstr, -1, str, strlen, NULL, NULL)
- #define ARRAYSIZE(A) (sizeof(A)/sizeof((A)[0]))
- #define ERR_BUFFER_SIZE 256
- #define DLL_STRINGIFY(s) #s
- #define DLL_LOAD_LIBRARY(name) LoadLibraryA(DLL_STRINGIFY(name))
- #define DLL_DECLARE_HANDLE(name) static HMODULE __dll_##name##_handle = NULL
- #define DLL_GET_HANDLE(name)
- #define DLL_FREE_HANDLE(name)
- #define DLL_DECLARE_FUNC_PREFIXNAME(api, ret, prefixname, name, args)
- #define DLL_DECLARE_FUNC(api, ret, name, args) DLL_DECLARE_FUNC_PREFIXNAME(api, ret, name, name, args)
- #define DLL_DECLARE_FUNC_PREFIXED(api, ret, prefix, name, args) DLL_DECLARE_FUNC_PREFIXNAME(api, ret, prefix##name, name, args)
- #define DLL_LOAD_FUNC_PREFIXNAME(dll, prefixname, name, ret_on_failure)
- #define DLL_LOAD_FUNC(dll, name, ret_on_failure) DLL_LOAD_FUNC_PREFIXNAME(dll, name, name, ret_on_failure)
- #define DLL_LOAD_FUNC_PREFIXED(dll, prefix, name, ret_on_failure) DLL_LOAD_FUNC_PREFIXNAME(dll, prefix##name, name, ret_on_failure)

34.259.1 Macro Definition Documentation

#define bool BOOL

#define true TRUE

#define false FALSE

#define safe_free(*p*) do {if (p != NULL) {free((void*)p); p = NULL;}} while(0)

#define safe_closehandle(*h*) do {if (h != INVALID_HANDLE_VALUE) {CloseHandle(h); h = INVALID_HANDLE_VALUE;}} while(0)

#define safe_min(*a, b*) MIN((size_t)(a), (size_t)(b))

#define safe_strcp(*dst, dst_max, src, count*)

Value:

```
do {memcpy(dst, src, safe_min(count, dst_max)); \
        ((char*)dst)[safe_min(count, dst_max)-1] = 0;} while(0)
```

#define safe_strcpy(*dst, dst_max, src*) safe_strcp(dst, dst_max, src, safe_strlen(src)+1)

#define safe_strncat(*dst, dst_max, src, count*) strncat(dst, src, safe_min(count, dst_max - safe_strlen(dst) - 1))

#define safe_strcat(*dst, dst_max, src*) safe_strncat(dst, dst_max, src, safe_strlen(src)+1)

#define safe_strcmp(*str1, str2*) strcmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2))

#define safe_stricmp(*str1, str2*) _stricmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2))

#define safe_strncmp(*str1, str2, count*) strncmp(((str1==NULL)?"<NULL>":str1), ((str2==NULL)?"<NULL>":str2), count)

#define safe_strlen(*str*) ((str==NULL)?0:strlen(str))

#define safe_sprintf(*dst, count, ...*) do {_snprintf(dst, count, __VA_ARGS__); (dst)[(count)-1] = 0; } while(0)

#define safe_stprintf _sntprintf

#define safe_tcslen(*str*) ((str==NULL)?0:_tcslen(str))

#define safe_unref_device(*dev*) do {if (dev != NULL) {libusb_unref_device(dev); dev = NULL;}} while(0)

#define wchar_to_utf8_ms(*wstr, str, strlen*) WideCharToMultiByte(CP_UTF8, 0, wstr, -1, str, strlen, NULL, NULL)

#define ARRAYSIZE(*A*) (sizeof(A)/sizeof((A)[0]))

#define ERR_BUFFER_SIZE 256

#define DLL_STRINGIFY(*s*) #s

#define DLL_LOAD_LIBRARY(*name*) LoadLibraryA(DLL_STRINGIFY(name))

#define DLL_DECLARE_HANDLE(*name*) static HMODULE __dll_##name##_handle = NULL

#define DLL_GET_HANDLE(*name*)

Value:

```
do {                                                    \
                __dll_##name##_handle = DLL_LOAD_LIBRARY(name); \
                if (!__dll_##name##_handle)                     \
                        return LIBUSB_ERROR_OTHER;              \
        } while (0)
```

#define DLL_FREE_HANDLE(*name*)

Value:

```
do {                                                    \
                if (__dll_##name##_handle) {                    \
                        FreeLibrary(__dll_##name##_handle);     \
                        __dll_##name##_handle = NULL;               \
                }                                               \
        } while(0)
```

#define DLL_DECLARE_FUNC_PREFIXNAME(*api, ret, prefixname, name, args*)

Value:

```
typedef ret (api * __dll_##name##_func_t)args;                  \
        static __dll_##name##_func_t prefixname = NULL
```

#define DLL_DECLARE_FUNC(*api, ret, name, args*) DLL_DECLARE_FUNC_PREFIXNAME(api, ret, name, name, args)

#define DLL_DECLARE_FUNC_PREFIXED(*api, ret, prefix, name, args*) DLL_DECLARE_FUNC_PREFIXNAME(api, ret, prefix##name, name, args)

#define DLL_LOAD_FUNC_PREFIXNAME(*dll, prefixname, name, ret_on_failure*)

Value:

```
do {                                                                    \
                HMODULE h = __dll_##dll##_handle;                       \
                prefixname = (__dll_##name##_func_t)GetProcAddress(h,   \
                                DLL_STRINGIFY(name));                       \
                if (prefixname)                                         \
                        break;                                          \
                prefixname = (__dll_##name##_func_t)GetProcAddress(h,   \
                                DLL_STRINGIFY(name) DLL_STRINGIFY(A));      \
                if (prefixname)                                         \
                        break;                                          \
                prefixname = (__dll_##name##_func_t)GetProcAddress(h,   \
                                DLL_STRINGIFY(name) DLL_STRINGIFY(W));      \
                if (prefixname)                                         \
                        break;                                          \
                if (ret_on_failure)                                     \
                        return LIBUSB_ERROR_NOT_FOUND;                    \
        } while(0)
```

#define DLL_LOAD_FUNC(*dll, name, ret_on_failure*) DLL_LOAD_FUNC_PREFIXNAME(dll, name, name, ret_on_failure)

#define DLL_LOAD_FUNC_PREFIXED(*dll, prefix, name, ret_on_failure*) DLL_LOAD_FUNC_PREFIXNAME(dll, prefix##name, name, ret_on_failure)

34.260 lib/ndkbuild/libusb/libusb/os/windows_nt_common.c File Reference

```
#include <config.h>
#include <inttypes.h>
#include <process.h>
#include <stdio.h>
#include "libusbi.h"
#include "windows_common.h"
#include "windows_nt_common.h"
```

Data Structures

- struct timer_request
- struct htab_entry

Macros

- #define TIMER_REQUEST_RETRY_MS 100
- #define WM_TIMER_REQUEST (WM_USER + 1)
- #define WM_TIMER_EXIT (WM_USER + 2)
- #define HTAB_SIZE 1021

Typedefs

- typedef struct htab_entry htab_entry

Functions

- DLL_DECLARE_HANDLE (User32)
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, GetMessageA,(LPMSG, HWND, UINT, UINT))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, PeekMessageA,(LPMSG, HWND, UINT, UINT, UINT))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, PostThreadMessageA,(DWORD, UINT, WPARAM, LPARAM))
- unsigned long htab_hash (const char *str)
- void windows_destroy_clock (void)
- int windows_clock_gettime (int clk_id, struct timespec *tp)
- void windows_handle_callback (struct usbi_transfer *itransfer, uint32_t io_result, uint32_t io_size)
- int windows_handle_events (struct libusb_context *ctx, struct pollfd *fds, POLL_NFDS_TYPE nfds, int num_ready)
- int windows_common_init (struct libusb_context *ctx)
- void windows_common_exit (void)

Variables

- const uint64_t epoch_time = UINT64_C(116444736000000000)

34.260.1 Macro Definition Documentation

#define TIMER_REQUEST_RETRY_MS 100

#define WM_TIMER_REQUEST (WM_USER + 1)

#define WM_TIMER_EXIT (WM_USER + 2)

#define HTAB_SIZE 1021

34.260.2 Typedef Documentation

typedef struct **htab_entry** **htab_entry**

34.260.3 Function Documentation

DLL_DECLARE_HANDLE (User32)

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , GetMessageA , (LPMSG, HWND, UINT, UINT))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , PeekMessageA , (LPMSG, HWND, UINT, UINT, UINT))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , PostThreadMessageA , (DWORD, UINT, WPARAM, LPARAM))

unsigned long htab_hash (**const** char ∗ *str*)

void windows_destroy_clock (**void**)

int windows_clock_gettime (int *clk_id,* struct **timespec** ∗ *tp*)

void windows_handle_callback (struct **usbi_transfer** ∗ *itransfer,* **uint32_t** *io_result,* **uint32_t** *io_size*)

int windows_handle_events (struct **libusb_context** ∗ *ctx,* struct **pollfd** ∗ *fds,* **POLL_NFDS_TYPE** *nfds,* int *num_ready*)

int windows_common_init (struct **libusb_context** ∗ *ctx*)

void windows_common_exit (**void**)

34.260.4 Variable Documentation

const uint64_t epoch_time = **UINT64_C**(116444736000000000)

34.261 lib/ndkbuild/libusb/libusb/os/windows_nt_common.h File Reference

Data Structures

- struct USB_CONFIGURATION_DESCRIPTOR

Macros

- #define FACILITY_SETUPAPI 15

Typedefs

- typedef struct USB_CONFIGURATION_DESCRIPTOR USB_CONFIGURATION_DESCRIPTOR
- typedef struct USB_CONFIGURATION_DESCRIPTOR ∗ PUSB_CONFIGURATION_DESCRIPTOR
- typedef struct libusb_device_descriptor USB_DEVICE_DESCRIPTOR
- typedef struct libusb_device_descriptor ∗ PUSB_DEVICE_DESCRIPTOR

Functions

- int windows_common_init (struct libusb_context ∗ctx)
- void windows_common_exit (void)

- unsigned long htab_hash (const char *str)
- int windows_clock_gettime (int clk_id, struct timespec *tp)
- void windows_clear_transfer_priv (struct usbi_transfer *itransfer)
- int windows_copy_transfer_data (struct usbi_transfer *itransfer, uint32_t io_size)
- struct winfd * windows_get_fd (struct usbi_transfer *transfer)
- void windows_get_overlapped_result (struct usbi_transfer *transfer, struct winfd *pollable_fd, DWORD *io_result, DWORD *io_size)
- void windows_handle_callback (struct usbi_transfer *itransfer, uint32_t io_result, uint32_t io_size)
- int windows_handle_events (struct libusb_context *ctx, struct pollfd *fds, POLL_NFDS_TYPE nfds, int num_ready)

34.261.1 Macro Definition Documentation

#define FACILITY_SETUPAPI 15

34.261.2 Typedef Documentation

typedef struct USB_CONFIGURATION_DESCRIPTOR USB_CONFIGURATION_DESCRIPTOR

typedef struct USB_CONFIGURATION_DESCRIPTOR * PUSB_CONFIGURATION_DESCRIPTOR

typedef struct libusb_device_descriptor USB_DEVICE_DESCRIPTOR

typedef struct libusb_device_descriptor * PUSB_DEVICE_DESCRIPTOR

34.261.3 Function Documentation

int windows_common_init (struct libusb_context * *ctx*)

void windows_common_exit (void)

unsigned long htab_hash (const char * *str*)

int windows_clock_gettime (int *clk_id*, struct timespec * *tp*)

void windows_clear_transfer_priv (struct usbi_transfer * *itransfer*)

int windows_copy_transfer_data (struct usbi_transfer * *itransfer*, uint32_t *io_size*)

struct winfd* windows_get_fd (struct usbi_transfer * *transfer*)

void windows_get_overlapped_result (struct usbi_transfer * *transfer*, struct winfd * *pollable_fd*, DWORD * *io_result*, DWORD * *io_size*)

void windows_handle_callback (struct usbi_transfer * *itransfer*, uint32_t *io_result*, uint32_t *io_size*)

int windows_handle_events (struct libusb_context * *ctx*, struct pollfd * *fds*, POLL_NFDS_TYPE *nfds*, int *num_ready*)

34.262 lib/ndkbuild/libusb/libusb/os/windows_usbdk.c File Reference

```
#include <config.h>
```

34.263 lib/ndkbuild/libusb/libusb/os/windows_usbdk.h File Reference

Data Structures

- struct tag_USB_DK_DEVICE_ID
- struct tag_USB_DK_DEVICE_INFO
- struct tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST
- struct tag_USB_DK_ISO_TRANSFER_RESULT
- struct tag_USB_DK_GEN_TRANSFER_RESULT
- struct tag_USB_DK_TRANSFER_RESULT
- struct tag_USB_DK_TRANSFER_REQUEST

Typedefs

- typedef struct tag_USB_DK_DEVICE_ID USB_DK_DEVICE_ID
- typedef struct tag_USB_DK_DEVICE_ID * PUSB_DK_DEVICE_ID
- typedef struct tag_USB_DK_DEVICE_INFO USB_DK_DEVICE_INFO
- typedef struct tag_USB_DK_DEVICE_INFO * PUSB_DK_DEVICE_INFO
- typedef struct tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST USB_DK_CONFIG_DESCRIPTOR_REQUEST
- typedef struct tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST * PUSB_DK_CONFIG_DESCRIPTOR_REQUEST
- typedef struct tag_USB_DK_ISO_TRANSFER_RESULT USB_DK_ISO_TRANSFER_RESULT
- typedef struct tag_USB_DK_ISO_TRANSFER_RESULT * PUSB_DK_ISO_TRANSFER_RESULT
- typedef struct tag_USB_DK_GEN_TRANSFER_RESULT USB_DK_GEN_TRANSFER_RESULT
- typedef struct tag_USB_DK_GEN_TRANSFER_RESULT * PUSB_DK_GEN_TRANSFER_RESULT
- typedef struct tag_USB_DK_TRANSFER_RESULT USB_DK_TRANSFER_RESULT
- typedef struct tag_USB_DK_TRANSFER_RESULT * PUSB_DK_TRANSFER_RESULT
- typedef struct tag_USB_DK_TRANSFER_REQUEST USB_DK_TRANSFER_REQUEST
- typedef struct tag_USB_DK_TRANSFER_REQUEST * PUSB_DK_TRANSFER_REQUEST
- typedef BOOL(__cdecl * USBDK_GET_DEVICES_LIST)(PUSB_DK_DEVICE_INFO *DeviceInfo, PULONG DeviceNumber)
- typedef void(__cdecl * USBDK_RELEASE_DEVICES_LIST)(PUSB_DK_DEVICE_INFO DeviceInfo)
- typedef HANDLE(__cdecl * USBDK_START_REDIRECT)(PUSB_DK_DEVICE_ID DeviceId)
- typedef BOOL(__cdecl * USBDK_STOP_REDIRECT)(HANDLE DeviceHandle)
- typedef BOOL(__cdecl * USBDK_GET_CONFIGURATION_DESCRIPTOR)(PUSB_DK_CONFIG_DESCRIPTOR_REQUEST Request, PUSB_CONFIGURATION_DESCRIPTOR *Descriptor, PULONG Length)
- typedef void(__cdecl * USBDK_RELEASE_CONFIGURATION_DESCRIPTOR)(PUSB_CONFIGURATION_DESCRIPTOR Descriptor)
- typedef TransferResult(__cdecl * USBDK_WRITE_PIPE)(HANDLE DeviceHandle, PUSB_DK_TRANSFER_REQUEST Request, LPOVERLAPPED lpOverlapped)
- typedef TransferResult(__cdecl * USBDK_READ_PIPE)(HANDLE DeviceHandle, PUSB_DK_TRANSFER_REQUEST Request, LPOVERLAPPED lpOverlapped)
- typedef BOOL(__cdecl * USBDK_ABORT_PIPE)(HANDLE DeviceHandle, ULONG64 PipeAddress)
- typedef BOOL(__cdecl * USBDK_RESET_PIPE)(HANDLE DeviceHandle, ULONG64 PipeAddress)
- typedef BOOL(__cdecl * USBDK_SET_ALTSETTING)(HANDLE DeviceHandle, ULONG64 InterfaceIdx, ULONG64 AltSettingIdx)
- typedef BOOL(__cdecl * USBDK_RESET_DEVICE)(HANDLE DeviceHandle)
- typedef HANDLE(__cdecl * USBDK_GET_REDIRECTOR_SYSTEM_HANDLE)(HANDLE DeviceHandle)

Enumerations

- enum TransferResult { TransferFailure = 0, TransferSuccess, TransferSuccessAsync }
- enum USB_DK_DEVICE_SPEED { NoSpeed = 0, LowSpeed, FullSpeed, HighSpeed, SuperSpeed }
- enum USB_DK_TRANSFER_TYPE { ControlTransferType, BulkTransferType, IntertuptTransferType, IsochronousTransferType }

34.263.1 Typedef Documentation

typedef struct tag_USB_DK_DEVICE_ID USB_DK_DEVICE_ID

typedef struct tag_USB_DK_DEVICE_ID * PUSB_DK_DEVICE_ID

typedef struct tag_USB_DK_DEVICE_INFO USB_DK_DEVICE_INFO

typedef struct tag_USB_DK_DEVICE_INFO * PUSB_DK_DEVICE_INFO

typedef struct tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST USB_DK_CONFIG_DESCRIPTOR_REQUEST

typedef struct tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST * PUSB_DK_CONFIG_DESCRIPTOR_REQUEST

typedef struct tag_USB_DK_ISO_TRANSFER_RESULT USB_DK_ISO_TRANSFER_RESULT

typedef struct tag_USB_DK_ISO_TRANSFER_RESULT * PUSB_DK_ISO_TRANSFER_RESULT

typedef struct tag_USB_DK_GEN_TRANSFER_RESULT USB_DK_GEN_TRANSFER_RESULT

typedef struct tag_USB_DK_GEN_TRANSFER_RESULT * PUSB_DK_GEN_TRANSFER_RESULT

typedef struct tag_USB_DK_TRANSFER_RESULT USB_DK_TRANSFER_RESULT

typedef struct tag_USB_DK_TRANSFER_RESULT * PUSB_DK_TRANSFER_RESULT

typedef struct tag_USB_DK_TRANSFER_REQUEST USB_DK_TRANSFER_REQUEST

typedef struct tag_USB_DK_TRANSFER_REQUEST * PUSB_DK_TRANSFER_REQUEST

typedef BOOL(__cdecl * USBDK_GET_DEVICES_LIST)(PUSB_DK_DEVICE_INFO *DeviceInfo, PULONG DeviceNumber)

typedef void(__cdecl * USBDK_RELEASE_DEVICES_LIST)(PUSB_DK_DEVICE_INFO DeviceInfo)

typedef HANDLE(__cdecl * USBDK_START_REDIRECT)(PUSB_DK_DEVICE_ID DeviceId)

typedef BOOL(__cdecl * USBDK_STOP_REDIRECT)(HANDLE DeviceHandle)

typedef BOOL(__cdecl * USBDK_GET_CONFIGURATION_DESCRIPTOR)(PUSB_DK_CONFIG_DESCRIPTOR_REQUEST Request, PUSB_CONFIGURATION_DESCRIPTOR *Descriptor, PULONG Length)

typedef void(__cdecl * USBDK_RELEASE_CONFIGURATION_DESCRIPTOR)(PUSB_CONFIGURATION_DESCRIPTOR Descriptor)

typedef TransferResult(__cdecl * USBDK_WRITE_PIPE)(HANDLE DeviceHandle, PUSB_DK_TRANSFER_REQUEST Request, LPOVERLAPPED lpOverlapped)

typedef TransferResult(__cdecl * USBDK_READ_PIPE)(HANDLE DeviceHandle, PUSB_DK_TRANSFER_REQUEST Request, LPOVERLAPPED lpOverlapped)

typedef BOOL(__cdecl * USBDK_ABORT_PIPE)(HANDLE DeviceHandle, ULONG64 PipeAddress)

typedef BOOL(__cdecl * USBDK_RESET_PIPE)(HANDLE DeviceHandle, ULONG64 PipeAddress)

typedef BOOL(__cdecl * USBDK_SET_ALTSETTING)(HANDLE DeviceHandle, ULONG64 InterfaceIdx, ULONG64 AltSettingIdx)

typedef BOOL(__cdecl * USBDK_RESET_DEVICE)(HANDLE DeviceHandle)

typedef HANDLE(__cdecl * USBDK_GET_REDIRECTOR_SYSTEM_HANDLE)(HANDLE DeviceHandle)

34.263.2 Enumeration Type Documentation

enum TransferResult

Enumerator

TransferFailure
TransferSuccess
TransferSuccessAsync

enum USB_DK_DEVICE_SPEED

Enumerator

NoSpeed
LowSpeed
FullSpeed
HighSpeed
SuperSpeed

enum USB_DK_TRANSFER_TYPE

Enumerator

ControlTransferType

BulkTransferType

IntertuptTransferType

IsochronousTransferType

34.264 lib/ndkbuild/libusb/libusb/os/windows_winusb.c File Reference

```
#include <config.h>
#include <windows.h>
#include <setupapi.h>
#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <process.h>
#include <stdio.h>
#include <inttypes.h>
#include <objbase.h>
#include <winioctl.h>
#include "libusbi.h"
#include "poll_windows.h"
#include "windows_winusb.h"
```

Macros

- #define HANDLE_VALID(h) (((h) != 0) && ((h) != INVALID_HANDLE_VALUE))
- #define LOOP_CHECK(fcall)
- #define LOOP_BREAK(err)
- #define CHECK_WINUSBX_AVAILABLE(sub_api)
- #define CHECK_HID_AVAILABLE
- #define MAX_ENUM_GUIDS 64
- #define HCD_PASS 0
- #define HUB_PASS 1
- #define GEN_PASS 2
- #define DEV_PASS 3
- #define HID_PASS 4
- #define WinUSBX_Set(fn)

Functions

- void windows_clear_transfer_priv (struct usbi_transfer *itransfer)
- int windows_copy_transfer_data (struct usbi_transfer *itransfer, uint32_t io_size)
- struct winfd * windows_get_fd (struct usbi_transfer *transfer)
- void windows_get_overlapped_result (struct usbi_transfer *transfer, struct winfd *pollable_fd, DWORD *io_result, DWORD *io_size)

Variables

- int windows_version = WINDOWS_UNDEFINED
- const struct usbi_os_backend windows_backend
- const struct windows_usb_api_backend usb_api_backend [USB_API_MAX]

34.264.1 Macro Definition Documentation

#define HANDLE_VALID(*h*) (((h) != 0) && ((h) != INVALID_HANDLE_VALUE))

#define LOOP_CHECK(*fcall*)

Value:

```
{                                       \
                r = fcall;                      \
                if (r != LIBUSB_SUCCESS)  \
                        continue;               \
        }
```

#define LOOP_BREAK(*err*)

Value:

```
{                                       \
                r = err;                        \
                continue;                       \
        }
```

#define CHECK_WINUSBX_AVAILABLE(*sub_api*)

Value:

```
do {                                            \
                if (sub_api == SUB_API_NOTSET)            \
                        sub_api = priv->sub_api;        \
                if (!WinUSBX[sub_api].initialized)      \
                        return LIBUSB_ERROR_ACCESS;  \
        } while(0)
```

#define CHECK_HID_AVAILABLE

Value:

```
do {                                            \
                if (!api_hid_available)                 \
                        return LIBUSB_ERROR_ACCESS;  \
        } while (0)
```

#define MAX_ENUM_GUIDS 64

#define HCD_PASS 0

#define HUB_PASS 1

#define GEN_PASS 2

#define DEV_PASS 3

#define HID_PASS 4

#define WinUSBX_Set(*fn*)

Value:

```
do {                                                                            \
                if (native_winusb)                                                      \
                        WinUSBX[i].fn = (WinUsb_##fn##_t)GetProcAddress(h, "WinUsb_" #fn);      \
                else                                                                    \
                        pLibK_GetProcAddress((PVOID *)&WinUSBX[i].fn, i, KUSB_FNID_##fn);       \
        } while (0)
```

34.264.2 Function Documentation

void windows_clear_transfer_priv (**struct usbi_transfer** ∗ *itransfer*)

int windows_copy_transfer_data (**struct usbi_transfer** ∗ *itransfer,* **uint32_t** *io_size*)

struct **winfd**∗ windows_get_fd (**struct usbi_transfer** ∗ *transfer*)

void windows_get_overlapped_result (**struct usbi_transfer** ∗ *transfer,* **struct winfd** ∗ *pollable_fd,* DWORD ∗ *io_result,* DWORD ∗ *io_size*)

34.264.3 Variable Documentation

int windows_version = WINDOWS_UNDEFINED

const struct usbi_os_backend windows_backend

const struct windows_usb_api_backend usb_api_backend[USB_API_MAX]

34.265 lib/ndkbuild/libusb/libusb/os/windows_winusb.h File Reference

```
#include "windows_common.h"
#include "windows_nt_common.h"
```

Data Structures

- struct windows_usb_api_backend
- struct libusb_hid_descriptor
- struct hid_device_priv
- struct windows_device_priv
- struct interface_handle_t
- struct windows_device_handle_priv
- struct windows_transfer_priv
- struct driver_lookup
- struct USB_INTERFACE_DESCRIPTOR
- struct USB_CONFIGURATION_DESCRIPTOR_SHORT
- struct USB_ENDPOINT_DESCRIPTOR
- struct USB_DESCRIPTOR_REQUEST
- struct USB_HUB_DESCRIPTOR
- struct USB_ROOT_HUB_NAME
- struct USB_ROOT_HUB_NAME_FIXED
- struct USB_NODE_CONNECTION_NAME
- struct USB_NODE_CONNECTION_NAME_FIXED
- struct USB_HUB_NAME_FIXED
- struct USB_HUB_INFORMATION
- struct USB_MI_PARENT_INFORMATION
- struct USB_NODE_INFORMATION
- struct USB_PIPE_INFO
- struct USB_NODE_CONNECTION_INFORMATION_EX
- struct USB_HUB_CAP_FLAGS
- struct USB_HUB_CAPABILITIES
- struct USB_HUB_CAPABILITIES_EX
- struct WINUSB_PIPE_INFORMATION
- struct WINUSB_SETUP_PACKET
- struct winusb_interface
- struct HIDD_ATTRIBUTES
- struct HIDP_CAPS

Macros

- #define SPDRP_ADDRESS 28
- #define SPDRP_INSTALL_STATE 34
- #define MAX_CTRL_BUFFER_LENGTH 4096
- #define MAX_USB_DEVICES 256
- #define MAX_USB_STRING_LENGTH 128
- #define MAX_HID_REPORT_SIZE 1024
- #define MAX_HID_DESCRIPTOR_SIZE 256
- #define MAX_GUID_STRING_LENGTH 40
- #define MAX_PATH_LENGTH 128
- #define MAX_KEY_LENGTH 256
- #define LIST_SEPARATOR ';'
- #define INTERFACE_CLAIMED ((HANDLE)(intptr_t)0xD1B5)
- #define LIBUSB_COMPLETED (LIBUSB_SUCCESS + 1)
- #define USB_API_UNSUPPORTED 0
- #define USB_API_HUB 1
- #define USB_API_COMPOSITE 2
- #define USB_API_WINUSBX 3
- #define USB_API_HID 4

- #define USB_API_MAX 5
- #define USB_API_SET (1 << USB_API_MAX)
- #define SUB_API_NOTSET -1
- #define SUB_API_LIBUSBK 0
- #define SUB_API_LIBUSB0 1
- #define SUB_API_WINUSB 2
- #define SUB_API_MAX 3
- #define WINUSBX_DRV_NAMES {"libusbK", "libusb0", "WinUSB"}
- #define PRINT_UNSUPPORTED_API(fname)
- #define LIBUSB_DT_HID_SIZE 9
- #define HID_MAX_CONFIG_DESC_SIZE
- #define HID_MAX_REPORT_SIZE 1024
- #define HID_IN_EP 0x81
- #define HID_OUT_EP 0x02
- #define LIBUSB_REQ_RECIPIENT(request_type) ((request_type) & 0x1F)
- #define LIBUSB_REQ_TYPE(request_type) ((request_type) & (0x03 << 5))
- #define LIBUSB_REQ_IN(request_type) ((request_type) & LIBUSB_ENDPOINT_IN)
- #define LIBUSB_REQ_OUT(request_type) (!LIBUSB_REQ_IN(request_type))
- #define HID_CTL_CODE(id) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_NEITHER, FILE_ANY_ACCESS)
- #define HID_BUFFER_CTL_CODE(id) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define HID_IN_CTL_CODE(id) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_IN_DIRECT, FILE_ANY_ACCESS)
- #define HID_OUT_CTL_CODE(id) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_OUT_DIRECT, FILE_ANY_ACCESS)
- #define IOCTL_HID_GET_FEATURE HID_OUT_CTL_CODE(100)
- #define IOCTL_HID_GET_INPUT_REPORT HID_OUT_CTL_CODE(104)
- #define IOCTL_HID_SET_FEATURE HID_IN_CTL_CODE(100)
- #define IOCTL_HID_SET_OUTPUT_REPORT HID_IN_CTL_CODE(101)
- #define CR_SUCCESS 0x00000000
- #define CR_NO_SUCH_DEVNODE 0x0000000D
- #define USB_DEVICE_DESCRIPTOR_TYPE LIBUSB_DT_DEVICE
- #define USB_CONFIGURATION_DESCRIPTOR_TYPE LIBUSB_DT_CONFIG
- #define USB_STRING_DESCRIPTOR_TYPE LIBUSB_DT_STRING
- #define USB_INTERFACE_DESCRIPTOR_TYPE LIBUSB_DT_INTERFACE
- #define USB_ENDPOINT_DESCRIPTOR_TYPE LIBUSB_DT_ENDPOINT
- #define USB_REQUEST_GET_STATUS LIBUSB_REQUEST_GET_STATUS
- #define USB_REQUEST_CLEAR_FEATURE LIBUSB_REQUEST_CLEAR_FEATURE
- #define USB_REQUEST_SET_FEATURE LIBUSB_REQUEST_SET_FEATURE
- #define USB_REQUEST_SET_ADDRESS LIBUSB_REQUEST_SET_ADDRESS
- #define USB_REQUEST_GET_DESCRIPTOR LIBUSB_REQUEST_GET_DESCRIPTOR
- #define USB_REQUEST_SET_DESCRIPTOR LIBUSB_REQUEST_SET_DESCRIPTOR
- #define USB_REQUEST_GET_CONFIGURATION LIBUSB_REQUEST_GET_CONFIGURATION
- #define USB_REQUEST_SET_CONFIGURATION LIBUSB_REQUEST_SET_CONFIGURATION
- #define USB_REQUEST_GET_INTERFACE LIBUSB_REQUEST_GET_INTERFACE
- #define USB_REQUEST_SET_INTERFACE LIBUSB_REQUEST_SET_INTERFACE
- #define USB_REQUEST_SYNC_FRAME LIBUSB_REQUEST_SYNCH_FRAME
- #define USB_GET_NODE_INFORMATION 258
- #define USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION 260
- #define USB_GET_NODE_CONNECTION_NAME 261
- #define USB_GET_HUB_CAPABILITIES 271
- #define USB_GET_NODE_CONNECTION_INFORMATION_EX 274
- #define USB_GET_HUB_CAPABILITIES_EX 276
- #define USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 279
- #define METHOD_BUFFERED 0
- #define FILE_ANY_ACCESS 0x00000000
- #define FILE_DEVICE_UNKNOWN 0x00000022
- #define FILE_DEVICE_USB FILE_DEVICE_UNKNOWN
- #define CTL_CODE(DeviceType, Function, Method, Access) (((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method))
- #define IOCTL_USB_GET_HUB_CAPABILITIES_EX CTL_CODE(FILE_DEVICE_USB, USB_GET_HUB_CAPABILITIES_EX, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_HUB_CAPABILITIES CTL_CODE(FILE_DEVICE_USB, USB_GET_HUB_CAPABILITIES, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION CTL_CODE(FILE_DEVICE_USB, USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, METHOD_BUFFERED, FILE_ANY_ACCESS)

- #define IOCTL_USB_GET_ROOT_HUB_NAME CTL_CODE(FILE_DEVICE_USB, HCD_GET_ROOT_HUB_NAME, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_NODE_INFORMATION CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_INFORMATION, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_INFORMATION_EX, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_INFORMATION_EX_V2, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_NODE_CONNECTION_ATTRIBUTES CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_ATTRIBUTES, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define IOCTL_USB_GET_NODE_CONNECTION_NAME CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_NAME, METHOD_BUFFERED, FILE_ANY_ACCESS)
- #define SHORT_PACKET_TERMINATE 0x01
- #define AUTO_CLEAR_STALL 0x02
- #define PIPE_TRANSFER_TIMEOUT 0x03
- #define IGNORE_SHORT_PACKETS 0x04
- #define ALLOW_PARTIAL_READS 0x05
- #define AUTO_FLUSH 0x06
- #define RAW_IO 0x07
- #define MAXIMUM_TRANSFER_SIZE 0x08
- #define AUTO_SUSPEND 0x81
- #define SUSPEND_DELAY 0x83
- #define DEVICE_SPEED 0x01
- #define LowSpeed 0x01
- #define FullSpeed 0x02
- #define HighSpeed 0x03
- #define HIDP_STATUS_SUCCESS 0x110000

Typedefs

- typedef DWORD DEVNODE
- typedef DWORD DEVINST
- typedef DEVNODE * PDEVNODE
- typedef DEVNODE * PDEVINST
- typedef DWORD RETURN_TYPE
- typedef RETURN_TYPE CONFIGRET
- typedef enum USB_CONNECTION_STATUS USB_CONNECTION_STATUS
- typedef enum USB_CONNECTION_STATUS * PUSB_CONNECTION_STATUS
- typedef enum USB_HUB_NODE USB_HUB_NODE
- typedef struct USB_INTERFACE_DESCRIPTOR USB_INTERFACE_DESCRIPTOR
- typedef struct USB_INTERFACE_DESCRIPTOR * PUSB_INTERFACE_DESCRIPTOR
- typedef struct USB_CONFIGURATION_DESCRIPTOR_SHORT USB_CONFIGURATION_DESCRIPTOR_SHORT
- typedef struct USB_ENDPOINT_DESCRIPTOR USB_ENDPOINT_DESCRIPTOR
- typedef struct USB_ENDPOINT_DESCRIPTOR * PUSB_ENDPOINT_DESCRIPTOR
- typedef struct USB_DESCRIPTOR_REQUEST USB_DESCRIPTOR_REQUEST
- typedef struct USB_DESCRIPTOR_REQUEST * PUSB_DESCRIPTOR_REQUEST
- typedef struct USB_HUB_DESCRIPTOR USB_HUB_DESCRIPTOR
- typedef struct USB_HUB_DESCRIPTOR * PUSB_HUB_DESCRIPTOR
- typedef struct USB_ROOT_HUB_NAME USB_ROOT_HUB_NAME
- typedef struct USB_ROOT_HUB_NAME * PUSB_ROOT_HUB_NAME
- typedef struct USB_ROOT_HUB_NAME_FIXED USB_ROOT_HUB_NAME_FIXED
- typedef struct USB_NODE_CONNECTION_NAME USB_NODE_CONNECTION_NAME
- typedef struct USB_NODE_CONNECTION_NAME * PUSB_NODE_CONNECTION_NAME
- typedef struct USB_NODE_CONNECTION_NAME_FIXED USB_NODE_CONNECTION_NAME_FIXED
- typedef struct USB_HUB_NAME_FIXED USB_HUB_NAME_FIXED
- typedef struct USB_HUB_INFORMATION USB_HUB_INFORMATION

- typedef struct USB_HUB_INFORMATION * PUSB_HUB_INFORMATION
- typedef struct USB_MI_PARENT_INFORMATION USB_MI_PARENT_INFORMATION
- typedef struct USB_MI_PARENT_INFORMATION * PUSB_MI_PARENT_INFORMATION
- typedef struct USB_NODE_INFORMATION USB_NODE_INFORMATION
- typedef struct USB_NODE_INFORMATION * PUSB_NODE_INFORMATION
- typedef struct USB_PIPE_INFO USB_PIPE_INFO
- typedef struct USB_PIPE_INFO * PUSB_PIPE_INFO
- typedef struct USB_NODE_CONNECTION_INFORMATION_EX USB_NODE_CONNECTION_INFORMATION_EX
- typedef struct USB_NODE_CONNECTION_INFORMATION_EX * PUSB_NODE_CONNECTION_INFORMATION_EX
- typedef union _USB_PROTOCOLS USB_PROTOCOLS
- typedef union _USB_PROTOCOLS * PUSB_PROTOCOLS
- typedef union _USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS
- typedef union _USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS * PUSB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS
- typedef struct _USB_NODE_CONNECTION_INFORMATION_EX_V2 USB_NODE_CONNECTION_INFORMATION_EX_V2
- typedef struct _USB_NODE_CONNECTION_INFORMATION_EX_V2 * PUSB_NODE_CONNECTION_INFORMATION_EX_V2
- typedef struct USB_HUB_CAP_FLAGS USB_HUB_CAP_FLAGS
- typedef struct USB_HUB_CAP_FLAGS * PUSB_HUB_CAP_FLAGS
- typedef struct USB_HUB_CAPABILITIES USB_HUB_CAPABILITIES
- typedef struct USB_HUB_CAPABILITIES * PUSB_HUB_CAPABILITIES
- typedef struct USB_HUB_CAPABILITIES_EX USB_HUB_CAPABILITIES_EX
- typedef struct USB_HUB_CAPABILITIES_EX * PUSB_HUB_CAPABILITIES_EX
- typedef enum USBD_PIPE_TYPE USBD_PIPE_TYPE
- typedef struct WINUSB_PIPE_INFORMATION * PWINUSB_PIPE_INFORMATION
- typedef struct WINUSB_SETUP_PACKET * PWINUSB_SETUP_PACKET
- typedef void * WINUSB_INTERFACE_HANDLE
- typedef void * PWINUSB_INTERFACE_HANDLE
- typedef BOOL(WINAPI * WinUsb_AbortPipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID)
- typedef BOOL(WINAPI * WinUsb_ControlTransfer_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, WINUSB_SETUP_PACKET SetupPacket, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, LPOVERLAPPED Overlapped)
- typedef BOOL(WINAPI * WinUsb_FlushPipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID)
- typedef BOOL(WINAPI * WinUsb_Free_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle)
- typedef BOOL(WINAPI * WinUsb_GetAssociatedInterface_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR AssociatedInterfaceIndex, PWINUSB_INTERFACE_HANDLE AssociatedInterfaceHandle)
- typedef BOOL(WINAPI * WinUsb_GetCurrentAlternateSetting_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, PUCHAR AlternateSetting)
- typedef BOOL(WINAPI * WinUsb_GetDescriptor_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR DescriptorType, UCHAR Index, USHORT LanguageID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred)
- typedef BOOL(WINAPI * WinUsb_GetOverlappedResult_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, LPOVERLAPPED lpOverlapped, LPDWORD lpNumberOfBytesTransferred, BOOL bWait)
- typedef BOOL(WINAPI * WinUsb_GetPipePolicy_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID, ULONG PolicyType, PULONG ValueLength, PVOID Value)
- typedef BOOL(WINAPI * WinUsb_GetPowerPolicy_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, ULONG PolicyType, PULONG ValueLength, PVOID Value)
- typedef BOOL(WINAPI * WinUsb_Initialize_t)(HANDLE DeviceHandle, PWINUSB_INTERFACE_HANDLE InterfaceHandle)
- typedef BOOL(WINAPI * WinUsb_QueryDeviceInformation_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, ULONG InformationType, PULONG BufferLength, PVOID Buffer)

- typedef BOOL(WINAPI * WinUsb_QueryInterfaceSettings_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR AlternateSettingNumber, PUSB_INTERFACE_DESCRIPTOR UsbAltInterfaceDescriptor)
- typedef BOOL(WINAPI * WinUsb_QueryPipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR AlternateInterfaceNumber, UCHAR PipeIndex, PWINUSB_PIPE_INFORMATION PipeInformation)
- typedef BOOL(WINAPI * WinUsb_ReadPipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, LPOVERLAPPED Overlapped)
- typedef BOOL(WINAPI * WinUsb_ResetPipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID)
- typedef BOOL(WINAPI * WinUsb_SetCurrentAlternateSetting_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR AlternateSetting)
- typedef BOOL(WINAPI * WinUsb_SetPipePolicy_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID, ULONG PolicyType, ULONG ValueLength, PVOID Value)
- typedef BOOL(WINAPI * WinUsb_SetPowerPolicy_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, ULONG PolicyType, ULONG ValueLength, PVOID Value)
- typedef BOOL(WINAPI * WinUsb_WritePipe_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle, UCHAR PipeID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, LPOVERLAPPED Overlapped)
- typedef BOOL(WINAPI * WinUsb_ResetDevice_t)(WINUSB_INTERFACE_HANDLE InterfaceHandle)
- typedef enum _KUSB_FNID KUSB_FNID
- typedef struct _KLIB_VERSION KLIB_VERSION
- typedef KLIB_VERSION * PKLIB_VERSION
- typedef BOOL(WINAPI * LibK_GetProcAddress_t)(PVOID *ProcAddress, ULONG DriverID, ULONG FunctionID)
- typedef VOID(WINAPI * LibK_GetVersion_t)(PKLIB_VERSION Version)
- typedef void * PHIDP_PREPARSED_DATA
- typedef struct HIDD_ATTRIBUTES * PHIDD_ATTRIBUTES
- typedef USHORT USAGE
- typedef struct HIDP_CAPS * PHIDP_CAPS
- typedef enum _HIDP_REPORT_TYPE HIDP_REPORT_TYPE
- typedef struct _HIDP_VALUE_CAPS HIDP_VALUE_CAPS
- typedef struct _HIDP_VALUE_CAPS * PHIDP_VALUE_CAPS

Enumerations

- enum libusb_hid_request_type {
  HID_REQ_GET_REPORT = 0x01, HID_REQ_GET_IDLE = 0x02, HID_REQ_GET_PROTOCOL = 0x03, HID_REQ_SET_REPORT = 0x09,
  HID_REQ_SET_IDLE = 0x0A, HID_REQ_SET_PROTOCOL = 0x0B }
- enum libusb_hid_report_type { HID_REPORT_TYPE_INPUT = 0x01, HID_REPORT_TYPE_OUTPUT = 0x02, HID_REPORT_TYPE_FEATURE = 0x03 }
- enum USB_CONNECTION_STATUS {
  NoDeviceConnected, DeviceConnected, DeviceFailedEnumeration, DeviceGeneralFailure,
  DeviceCausedOvercurrent, DeviceNotEnoughPower, DeviceNotEnoughBandwidth, DeviceHubNestedTooDeeply,
  DeviceInLegacyHub }
- enum USB_HUB_NODE { UsbHub, UsbMIParent }
- enum USBD_PIPE_TYPE { UsbdPipeTypeControl, UsbdPipeTypeIsochronous, UsbdPipeTypeBulk, UsbdPipeTypeInterrupt }

Functions

- DLL_DECLARE_HANDLE (OLE32)
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, HRESULT, p, CLSIDFromString,(LPCOLESTR, LPCLSID))
- DLL_DECLARE_HANDLE (Kernel32)
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, IsWow64Process,(HANDLE, PBOOL))
- DLL_DECLARE_HANDLE (SetupAPI)
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, HDEVINFO, p, SetupDiGetClassDevsA,(const GUID *, PCSTR, HWND, DWORD))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, SetupDiEnumDeviceInfo,(HDEVINFO, DWORD, PSP_DEVINFO_DATA))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, SetupDiEnumDeviceInterfaces,(HDEVINFO, PSP_DEVINFO_DATA, const GUID *, DWORD, PSP_DEVICE_INTERFACE_DATA))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, SetupDiGetDeviceInterfaceDetailA,(HDEVINFO, PSP_DEVICE_INTERFACE_DATA, PSP_DEVICE_INTERFACE_DETAIL_DATA_A, DWORD, PDWORD, PSP_DEVINFO_DATA))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, SetupDiDestroyDeviceInfoList,(HDEVINFO))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, HKEY, p, SetupDiOpenDevRegKey,(HDEVINFO, PSP_DEVINFO_DATA, DWORD, DWORD, DWORD, REGSAM))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, BOOL, p, SetupDiGetDeviceRegistryPropertyA,(HDEVINFO, PSP_DEVINFO_DATA, DWORD, PDWORD, PBYTE, DWORD, PDWORD))

- DLL_DECLARE_FUNC_PREFIXED (WINAPI, HKEY, p, SetupDiOpenDeviceInterfaceRegKey,(HDEVINFO, PSP_DEVICE_INTERFACE_DATA, DWORD, DWORD))
- DLL_DECLARE_HANDLE (AdvAPI32)
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, LONG, p, RegQueryValueExW,(HKEY, LPCWSTR, LPDWORD, LPDWORD, LPBYTE, LPDWORD))
- DLL_DECLARE_FUNC_PREFIXED (WINAPI, LONG, p, RegCloseKey,(HKEY))
- DLL_DECLARE_HANDLE (Cfgmgr32)
- DLL_DECLARE_FUNC (WINAPI, CONFIGRET, CM_Get_Parent,(PDEVINST, DEVINST, ULONG))
- DLL_DECLARE_FUNC (WINAPI, CONFIGRET, CM_Get_Child,(PDEVINST, DEVINST, ULONG))
- DLL_DECLARE_FUNC (WINAPI, CONFIGRET, CM_Get_Sibling,(PDEVINST, DEVINST, ULONG))
- DLL_DECLARE_FUNC (WINAPI, CONFIGRET, CM_Get_Device_IDA,(DEVINST, PCHAR, ULONG, ULONG))
- DLL_DECLARE_HANDLE (hid)
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetAttributes,(HANDLE, PHIDD_ATTRIBUTES))
- DLL_DECLARE_FUNC (WINAPI, VOID, HidD_GetHidGuid,(LPGUID))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetPreparsedData,(HANDLE, PHIDP_PREPARSED_DATA *))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_FreePreparsedData,(PHIDP_PREPARSED_DATA))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetManufacturerString,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetProductString,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetSerialNumberString,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, LONG, HidP_GetCaps,(PHIDP_PREPARSED_DATA, PHIDP_CAPS))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_SetNumInputBuffers,(HANDLE, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_SetFeature,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetFeature,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetPhysicalDescriptor,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_GetInputReport,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_SetOutputReport,(HANDLE, PVOID, ULONG))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidD_FlushQueue,(HANDLE))
- DLL_DECLARE_FUNC (WINAPI, BOOL, HidP_GetValueCaps,(HIDP_REPORT_TYPE, PHIDP_VALUE_CAPS, PULONG, PHIDP_PREPARSED_DATA))

Variables

- const GUID GUID_DEVINTERFACE_USB_HOST_CONTROLLER = { 0x3ABF6F2D, 0x71C4, 0x462A, {0x8A, 0x92, 0x1E, 0x68, 0x61, 0xE6, 0xAF, 0x27} }
- const GUID GUID_DEVINTERFACE_USB_DEVICE = { 0xA5DCBF10, 0x6530, 0x11D2, {0x90, 0x1F, 0x00, 0xC0, 0x4F, 0xB9, 0x51, 0xED} }
- const GUID GUID_DEVINTERFACE_USB_HUB = { 0xF18A0E88, 0xC30C, 0x11D0, {0x88, 0x15, 0x00, 0xA0, 0xC9, 0x06, 0xBE, 0xD8} }
- const GUID GUID_DEVINTERFACE_LIBUSB0_FILTER = { 0xF9F3FF14, 0xAE21, 0x48A0, {0x8A, 0x25, 0x80, 0x11, 0xA7, 0xA9, 0x31, 0xD9} }
- const struct windows_usb_api_backend usb_api_backend [USB_API_MAX]

34.265.1 Macro Definition Documentation

#define SPDRP_ADDRESS 28

#define SPDRP_INSTALL_STATE 34

#define MAX_CTRL_BUFFER_LENGTH 4096

#define MAX_USB_DEVICES 256

#define MAX_USB_STRING_LENGTH 128

#define MAX_HID_REPORT_SIZE 1024

#define MAX_HID_DESCRIPTOR_SIZE 256

#define MAX_GUID_STRING_LENGTH 40

#define MAX_PATH_LENGTH 128

#define MAX_KEY_LENGTH 256

#define LIST_SEPARATOR ';'

#define INTERFACE_CLAIMED ((HANDLE)(intptr_t)0xD1B5)

#define LIBUSB_COMPLETED (LIBUSB_SUCCESS + 1)

#define USB_API_UNSUPPORTED 0

#define USB_API_HUB 1

#define USB_API_COMPOSITE 2

#define USB_API_WINUSBX 3

#define USB_API_HID 4

#define USB_API_MAX 5

#define USB_API_SET (1 << USB_API_MAX)

#define SUB_API_NOTSET -1

#define SUB_API_LIBUSBK 0

#define SUB_API_LIBUSB0 1

#define SUB_API_WINUSB 2

#define SUB_API_MAX 3

#define WINUSBX_DRV_NAMES {"libusbK", "libusb0", "WinUSB"}

#define PRINT_UNSUPPORTED_API(*fname*)

Value:

```
usbi_dbg("unsupported API call for '"                    \
                #fname "' (unrecognized device driver)");       \
        return LIBUSB_ERROR_NOT_SUPPORTED;
```

#define LIBUSB_DT_HID_SIZE 9

#define HID_MAX_CONFIG_DESC_SIZE

Value:

```
(LIBUSB_DT_CONFIG_SIZE + LIBUSB_DT_INTERFACE_SIZE \
        + LIBUSB_DT_HID_SIZE + 2 * LIBUSB_DT_ENDPOINT_SIZE)
```

#define HID_MAX_REPORT_SIZE 1024

#define HID_IN_EP 0x81

#define HID_OUT_EP 0x02

#define LIBUSB_REQ_RECIPIENT(*request_type*) ((request_type) & 0x1F)

#define LIBUSB_REQ_TYPE(*request_type*) ((request_type) & (0x03 << 5))

#define LIBUSB_REQ_IN(*request_type*) ((request_type) & LIBUSB_ENDPOINT_IN)

#define LIBUSB_REQ_OUT(*request_type*) (!LIBUSB_REQ_IN(request_type))

#define HID_CTL_CODE(*id*) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_NEITHER, FILE_ANY_ACCESS)

#define HID_BUFFER_CTL_CODE(*id*) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_BUFFERED, FILE_ANY_ACCESS)

#define HID_IN_CTL_CODE(*id*) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_IN_DIRECT, FILE_ANY_ACCESS)

#define HID_OUT_CTL_CODE(*id*) CTL_CODE (FILE_DEVICE_KEYBOARD, (id), METHOD_OUT_DIRECT, FILE_ANY_ACCESS)

#define IOCTL_HID_GET_FEATURE HID_OUT_CTL_CODE(100)

#define IOCTL_HID_GET_INPUT_REPORT HID_OUT_CTL_CODE(104)

#define IOCTL_HID_SET_FEATURE HID_IN_CTL_CODE(100)

#define IOCTL_HID_SET_OUTPUT_REPORT HID_IN_CTL_CODE(101)

#define CR_SUCCESS 0x00000000

#define CR_NO_SUCH_DEVNODE 0x0000000D

#define USB_DEVICE_DESCRIPTOR_TYPE LIBUSB_DT_DEVICE

#define USB_CONFIGURATION_DESCRIPTOR_TYPE LIBUSB_DT_CONFIG

#define USB_STRING_DESCRIPTOR_TYPE LIBUSB_DT_STRING

#define USB_INTERFACE_DESCRIPTOR_TYPE LIBUSB_DT_INTERFACE

#define USB_ENDPOINT_DESCRIPTOR_TYPE LIBUSB_DT_ENDPOINT

#define USB_REQUEST_GET_STATUS LIBUSB_REQUEST_GET_STATUS

#define USB_REQUEST_CLEAR_FEATURE LIBUSB_REQUEST_CLEAR_FEATURE

#define USB_REQUEST_SET_FEATURE LIBUSB_REQUEST_SET_FEATURE

#define USB_REQUEST_SET_ADDRESS LIBUSB_REQUEST_SET_ADDRESS

#define USB_REQUEST_GET_DESCRIPTOR LIBUSB_REQUEST_GET_DESCRIPTOR

#define USB_REQUEST_SET_DESCRIPTOR LIBUSB_REQUEST_SET_DESCRIPTOR

#define USB_REQUEST_GET_CONFIGURATION LIBUSB_REQUEST_GET_CONFIGURATION

#define USB_REQUEST_SET_CONFIGURATION LIBUSB_REQUEST_SET_CONFIGURATION

#define USB_REQUEST_GET_INTERFACE LIBUSB_REQUEST_GET_INTERFACE

#define USB_REQUEST_SET_INTERFACE LIBUSB_REQUEST_SET_INTERFACE

#define USB_REQUEST_SYNC_FRAME LIBUSB_REQUEST_SYNCH_FRAME

#define USB_GET_NODE_INFORMATION 258

#define USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION 260

#define USB_GET_NODE_CONNECTION_NAME 261

#define USB_GET_HUB_CAPABILITIES 271

#define USB_GET_NODE_CONNECTION_INFORMATION_EX 274

#define USB_GET_HUB_CAPABILITIES_EX 276

#define USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 279

#define METHOD_BUFFERED 0

#define FILE_ANY_ACCESS 0x00000000

#define FILE_DEVICE_UNKNOWN 0x00000022

#define FILE_DEVICE_USB FILE_DEVICE_UNKNOWN

#define CTL_CODE(*DeviceType, Function, Method, Access*) (((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method))

#define IOCTL_USB_GET_HUB_CAPABILITIES_EX CTL_CODE(FILE_DEVICE_USB, USB_GET_HUB_CAPABILITIES_EX, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_HUB_CAPABILITIES CTL_CODE(FILE_DEVICE_USB, USB_GET_HUB_CAPABILITIES, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION CTL_CODE(FILE_DEVICE_USB, USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_ROOT_HUB_NAME CTL_CODE(FILE_DEVICE_USB, HCD_GET_ROOT_HUB_NAME, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_NODE_INFORMATION CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_INFORMATION, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_INFORMATION_EX, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_NODE_CONNECTION_INFORMATION_EX_V2 CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_INFORMATION_EX_V2, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_NODE_CONNECTION_ATTRIBUTES CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_ATTRIBUTES, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_USB_GET_NODE_CONNECTION_NAME CTL_CODE(FILE_DEVICE_USB, USB_GET_NODE_CONNECTION_NAME, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define SHORT_PACKET_TERMINATE 0x01

#define AUTO_CLEAR_STALL 0x02

#define PIPE_TRANSFER_TIMEOUT 0x03

#define IGNORE_SHORT_PACKETS 0x04

#define ALLOW_PARTIAL_READS 0x05

#define AUTO_FLUSH 0x06

#define RAW_IO 0x07

#define MAXIMUM_TRANSFER_SIZE 0x08

#define AUTO_SUSPEND 0x81

#define SUSPEND_DELAY 0x83

#define DEVICE_SPEED 0x01

#define LowSpeed 0x01

#define FullSpeed 0x02

#define HighSpeed 0x03

#define HIDP_STATUS_SUCCESS 0x110000

34.265.2 Typedef Documentation

typedef DWORD DEVNODE

typedef DWORD DEVINST

typedef DEVNODE* PDEVNODE

typedef DEVNODE * PDEVINST

typedef DWORD **RETURN_TYPE**

typedef **RETURN_TYPE CONFIGRET**

typedef enum **USB_CONNECTION_STATUS USB_CONNECTION_STATUS**

typedef enum **USB_CONNECTION_STATUS * PUSB_CONNECTION_STATUS**

typedef enum **USB_HUB_NODE USB_HUB_NODE**

typedef struct **USB_INTERFACE_DESCRIPTOR USB_INTERFACE_DESCRIPTOR**

typedef struct **USB_INTERFACE_DESCRIPTOR * PUSB_INTERFACE_DESCRIPTOR**

typedef struct **USB_CONFIGURATION_DESCRIPTOR_SHORT USB_CONFIGURATION_DESCRIPTOR_SHORT**

typedef struct **USB_ENDPOINT_DESCRIPTOR USB_ENDPOINT_DESCRIPTOR**

typedef struct **USB_ENDPOINT_DESCRIPTOR * PUSB_ENDPOINT_DESCRIPTOR**

typedef struct **USB_DESCRIPTOR_REQUEST USB_DESCRIPTOR_REQUEST**

typedef struct **USB_DESCRIPTOR_REQUEST * PUSB_DESCRIPTOR_REQUEST**

typedef struct **USB_HUB_DESCRIPTOR USB_HUB_DESCRIPTOR**

typedef struct **USB_HUB_DESCRIPTOR * PUSB_HUB_DESCRIPTOR**

typedef struct **USB_ROOT_HUB_NAME USB_ROOT_HUB_NAME**

typedef struct **USB_ROOT_HUB_NAME * PUSB_ROOT_HUB_NAME**

typedef struct **USB_ROOT_HUB_NAME_FIXED USB_ROOT_HUB_NAME_FIXED**

typedef struct **USB_NODE_CONNECTION_NAME USB_NODE_CONNECTION_NAME**

typedef struct **USB_NODE_CONNECTION_NAME * PUSB_NODE_CONNECTION_NAME**

typedef struct **USB_NODE_CONNECTION_NAME_FIXED USB_NODE_CONNECTION_NAME_FIXED**

typedef struct **USB_HUB_NAME_FIXED USB_HUB_NAME_FIXED**

typedef struct **USB_HUB_INFORMATION USB_HUB_INFORMATION**

typedef struct **USB_HUB_INFORMATION * PUSB_HUB_INFORMATION**

typedef struct **USB_MI_PARENT_INFORMATION USB_MI_PARENT_INFORMATION**

typedef struct **USB_MI_PARENT_INFORMATION * PUSB_MI_PARENT_INFORMATION**

typedef struct **USB_NODE_INFORMATION USB_NODE_INFORMATION**

typedef struct **USB_NODE_INFORMATION * PUSB_NODE_INFORMATION**

typedef struct **USB_PIPE_INFO USB_PIPE_INFO**

typedef struct **USB_PIPE_INFO * PUSB_PIPE_INFO**

typedef struct **USB_NODE_CONNECTION_INFORMATION_EX USB_NODE_CONNECTION_INFORMATION_EX**

typedef struct **USB_NODE_CONNECTION_INFORMATION_EX * PUSB_NODE_CONNECTION_INFORMATION_EX**

typedef union _USB_PROTOCOLS **USB_PROTOCOLS**

typedef union _USB_PROTOCOLS * **PUSB_PROTOCOLS**

typedef union _USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS **USB_NODE_CONNECTION_INFORMATION_EX-_V2_FLAGS**

typedef union _USB_NODE_CONNECTION_INFORMATION_EX_V2_FLAGS * **PUSB_NODE_CONNECTION_INFORMATION_-EX_V2_FLAGS**

typedef struct _USB_NODE_CONNECTION_INFORMATION_EX_V2 **USB_NODE_CONNECTION_INFORMATION_EX_V2**

typedef struct _USB_NODE_CONNECTION_INFORMATION_EX_V2 * **PUSB_NODE_CONNECTION_INFORMATION_EX_V2**

typedef struct **USB_HUB_CAP_FLAGS** **USB_HUB_CAP_FLAGS**

typedef struct **USB_HUB_CAP_FLAGS** * **PUSB_HUB_CAP_FLAGS**

typedef struct **USB_HUB_CAPABILITIES** **USB_HUB_CAPABILITIES**

typedef struct **USB_HUB_CAPABILITIES** * **PUSB_HUB_CAPABILITIES**

typedef struct **USB_HUB_CAPABILITIES_EX** **USB_HUB_CAPABILITIES_EX**

typedef struct **USB_HUB_CAPABILITIES_EX** * **PUSB_HUB_CAPABILITIES_EX**

typedef enum **USBD_PIPE_TYPE** **USBD_PIPE_TYPE**

typedef struct **WINUSB_PIPE_INFORMATION** * **PWINUSB_PIPE_INFORMATION**

typedef struct **WINUSB_SETUP_PACKET** * **PWINUSB_SETUP_PACKET**

typedef **void*** **WINUSB_INTERFACE_HANDLE**

typedef **void** * **PWINUSB_INTERFACE_HANDLE**

typedef BOOL(WINAPI * WinUsb_AbortPipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID)

typedef BOOL(WINAPI * WinUsb_ControlTransfer_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, **WINUSB_SETUP_PACKET** SetupPacket, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, **LPOVERLAPPED** Overlapped)

typedef BOOL(WINAPI * WinUsb_FlushPipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID)

typedef BOOL(WINAPI * WinUsb_Free_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle)

typedef BOOL(WINAPI * WinUsb_GetAssociatedInterface_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR AssociatedInterfaceIndex, **PWINUSB_INTERFACE_HANDLE** AssociatedInterfaceHandle)

typedef BOOL(WINAPI * WinUsb_GetCurrentAlternateSetting_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, PUCHAR AlternateSetting)

typedef BOOL(WINAPI * WinUsb_GetDescriptor_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR DescriptorType, UCHAR Index, USHORT LanguageID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred)

typedef BOOL(WINAPI * WinUsb_GetOverlappedResult_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, **LPOVERLAPPED** lpOverlapped, LPDWORD lpNumberOfBytesTransferred, BOOL bWait)

typedef BOOL(WINAPI * WinUsb_GetPipePolicy_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID, ULONG PolicyType, PULONG ValueLength, PVOID Value)

typedef BOOL(WINAPI * WinUsb_GetPowerPolicy_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, ULONG PolicyType, PULONG ValueLength, PVOID Value)

typedef BOOL(WINAPI * WinUsb_Initialize_t)(HANDLE DeviceHandle, **PWINUSB_INTERFACE_HANDLE** InterfaceHandle)

typedef BOOL(WINAPI * WinUsb_QueryDeviceInformation_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, ULONG InformationType, PULONG BufferLength, PVOID Buffer)

typedef BOOL(WINAPI * WinUsb_QueryInterfaceSettings_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR AlternateSettingNumber, **PUSB_INTERFACE_DESCRIPTOR** UsbAltInterfaceDescriptor)

typedef BOOL(WINAPI * WinUsb_QueryPipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR AlternateInterfaceNumber, UCHAR PipeIndex, **PWINUSB_PIPE_INFORMATION** PipeInformation)

typedef BOOL(WINAPI * WinUsb_ReadPipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, **LPOVERLAPPED** Overlapped)

typedef BOOL(WINAPI * WinUsb_ResetPipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID)

typedef BOOL(WINAPI * WinUsb_SetCurrentAlternateSetting_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR AlternateSetting)

typedef BOOL(WINAPI * WinUsb_SetPipePolicy_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID, ULONG PolicyType, ULONG ValueLength, PVOID Value)

typedef BOOL(WINAPI * WinUsb_SetPowerPolicy_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, ULONG PolicyType, ULONG ValueLength, PVOID Value)

typedef BOOL(WINAPI * WinUsb_WritePipe_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle, UCHAR PipeID, PUCHAR Buffer, ULONG BufferLength, PULONG LengthTransferred, **LPOVERLAPPED** Overlapped)

typedef BOOL(WINAPI * WinUsb_ResetDevice_t)(**WINUSB_INTERFACE_HANDLE** InterfaceHandle)

typedef enum _KUSB_FNID **KUSB_FNID**

typedef struct _KLIB_VERSION **KLIB_VERSION**

typedef **KLIB_VERSION*** **PKLIB_VERSION**

typedef BOOL(WINAPI * LibK_GetProcAddress_t)(PVOID *ProcAddress, ULONG DriverID, ULONG FunctionID)

typedef VOID(WINAPI * LibK_GetVersion_t)(**PKLIB_VERSION** Version)

typedef **void*** **PHIDP_PREPARSED_DATA**

typedef struct **HIDD_ATTRIBUTES** * **PHIDD_ATTRIBUTES**

typedef USHORT **USAGE**

typedef struct **HIDP_CAPS** * **PHIDP_CAPS**

typedef enum _HIDP_REPORT_TYPE **HIDP_REPORT_TYPE**

typedef struct _HIDP_VALUE_CAPS **HIDP_VALUE_CAPS**

typedef struct _HIDP_VALUE_CAPS * **PHIDP_VALUE_CAPS**

34.265.3 Enumeration Type Documentation

enum **libusb_hid_request_type**

Enumerator

HID_REQ_GET_REPORT
HID_REQ_GET_IDLE
HID_REQ_GET_PROTOCOL
HID_REQ_SET_REPORT
HID_REQ_SET_IDLE
HID_REQ_SET_PROTOCOL

enum **libusb_hid_report_type**

Enumerator

HID_REPORT_TYPE_INPUT
HID_REPORT_TYPE_OUTPUT
HID_REPORT_TYPE_FEATURE

enum **USB_CONNECTION_STATUS**

Enumerator

NoDeviceConnected
DeviceConnected

DeviceFailedEnumeration

DeviceGeneralFailure

DeviceCausedOvercurrent

DeviceNotEnoughPower

DeviceNotEnoughBandwidth

DeviceHubNestedTooDeeply

DeviceInLegacyHub

enum **USB_HUB_NODE**

Enumerator

UsbHub

UsbMIParent

enum **USBD_PIPE_TYPE**

Enumerator

UsbdPipeTypeControl

UsbdPipeTypeIsochronous

UsbdPipeTypeBulk

UsbdPipeTypeInterrupt

34.265.4 Function Documentation

DLL_DECLARE_HANDLE (OLE32)

DLL_DECLARE_FUNC_PREFIXED (WINAPI , HRESULT , p , CLSIDFromString , (LPCOLESTR, LPCLSID))

DLL_DECLARE_HANDLE (Kernel32)

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , IsWow64Process , (HANDLE, PBOOL))

DLL_DECLARE_HANDLE (SetupAPI)

DLL_DECLARE_FUNC_PREFIXED (WINAPI , HDEVINFO , p , SetupDiGetClassDevsA , (const GUID *, PCSTR, HWND, DWORD))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , SetupDiEnumDeviceInfo , (HDEVINFO, DWORD, PSP_DEVINFO_DATA))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , SetupDiEnumDeviceInterfaces , (HDEVINFO, PSP_DEVINFO_DATA, const GUID *, DWORD, PSP_DEVICE_INTERFACE_DATA))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , SetupDiGetDeviceInterfaceDetailA , (HDEVINFO, PSP_DEVICE_INTERFACE_DATA, PSP_DEVICE_INTERFACE_DETAIL_DATA_A, DWORD, PDWORD, PSP_DEVINFO_DATA))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , SetupDiDestroyDeviceInfoList , (HDEVINFO))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , HKEY , p , SetupDiOpenDevRegKey , (HDEVINFO, PSP_DEVINFO_DATA, DWORD, DWORD, DWORD, REGSAM))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , BOOL , p , SetupDiGetDeviceRegistryPropertyA , (HDEVINFO, PSP_DEVINFO_DATA, DWORD, PDWORD, PBYTE, DWORD, PDWORD))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , HKEY , p , SetupDiOpenDeviceInterfaceRegKey , (HDEVINFO, PSP_DEVICE_INTERFACE_DATA, DWORD, DWORD))

DLL_DECLARE_HANDLE (AdvAPI32)

DLL_DECLARE_FUNC_PREFIXED (WINAPI , LONG , p , RegQueryValueExW , (HKEY, LPCWSTR, LPDWORD, LPDWORD, LPBYTE, LPDWORD))

DLL_DECLARE_FUNC_PREFIXED (WINAPI , LONG , p , RegCloseKey , (HKEY))

DLL_DECLARE_HANDLE (Cfgmgr32)

DLL_DECLARE_FUNC (WINAPI , CONFIGRET , CM_Get_Parent , (PDEVINST, DEVINST, ULONG))

DLL_DECLARE_FUNC (WINAPI , CONFIGRET , CM_Get_Child , (PDEVINST, DEVINST, ULONG))

DLL_DECLARE_FUNC (WINAPI , CONFIGRET , CM_Get_Sibling , (PDEVINST, DEVINST, ULONG))

DLL_DECLARE_FUNC (WINAPI , CONFIGRET , CM_Get_Device_IDA , (DEVINST, PCHAR, ULONG, ULONG))

DLL_DECLARE_HANDLE (hid)

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetAttributes , (HANDLE, PHIDD_ATTRIBUTES))

DLL_DECLARE_FUNC (WINAPI , VOID , HidD_GetHidGuid , (LPGUID))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetPreparsedData , (HANDLE, PHIDP_PREPARSED_DATA *))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_FreePreparsedData , (PHIDP_PREPARSED_DATA))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetManufacturerString , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetProductString , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetSerialNumberString , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , LONG , HidP_GetCaps , (PHIDP_PREPARSED_DATA, PHIDP_CAPS))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_SetNumInputBuffers , (HANDLE, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_SetFeature , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetFeature , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetPhysicalDescriptor , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_GetInputReport , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_SetOutputReport , (HANDLE, PVOID, ULONG))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidD_FlushQueue , (HANDLE))

DLL_DECLARE_FUNC (WINAPI , BOOL , HidP_GetValueCaps , (HIDP_REPORT_TYPE, PHIDP_VALUE_CAPS, PULONG, PHIDP_PREPARSED_DATA))

34.265.5 Variable Documentation

const GUID GUID_DEVINTERFACE_USB_HOST_CONTROLLER = { 0x3ABF6F2D, 0x71C4, 0x462A, {0x8A, 0x92, 0x1E, 0x68, 0x61, 0xE6, 0xAF, 0x27} }

const GUID GUID_DEVINTERFACE_USB_DEVICE = { 0xA5DCBF10, 0x6530, 0x11D2, {0x90, 0x1F, 0x00, 0xC0, 0x4F, 0xB9, 0x51, 0xED} }

const GUID GUID_DEVINTERFACE_USB_HUB = { 0xF18A0E88, 0xC30C, 0x11D0, {0x88, 0x15, 0x00, 0xA0, 0xC9, 0x06, 0xBE, 0xD8} }

const GUID GUID_DEVINTERFACE_LIBUSB0_FILTER = { 0xF9F3FF14, 0xAE21, 0x48A0, {0x8A, 0x25, 0x80, 0x11, 0xA7, 0xA9, 0x31, 0xD9} }

const struct windows_usb_api_backend usb_api_backend[USB_API_MAX]

34.266 lib/ndkbuild/libusb/libusb/strerror.c File Reference

```
#include <config.h>
#include <locale.h>
#include <stdlib.h>
#include <string.h>
#include "libusbi.h"
```

Functions

- int API_EXPORTED libusb_setlocale (const char *locale)

  Set the language, and only the language, not the encoding! used for translatable libusb messages.
- DEFAULT_VISIBILITY const char *LIBUSB_CALL libusb_strerror (enum libusb_error errcode)

  Returns a constant string with a short description of the given error code, this description is intended for displaying to the end user and will be in the language set by libusb_setlocale().

34.267 lib/ndkbuild/libusb/libusb/sync.c File Reference

```
#include <config.h>
#include <errno.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "libusbi.h"
```

Functions

- int API_EXPORTED libusb_control_transfer (libusb_device_handle *dev_handle, uint8_t bmRequestType, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char *data, uint16_t wLength, unsigned int timeout)

  Perform a USB control transfer.
- int API_EXPORTED libusb_bulk_transfer (struct libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *transferred, unsigned int timeout)

  Perform a USB bulk transfer.
- int API_EXPORTED libusb_interrupt_transfer (struct libusb_device_handle *dev_handle, unsigned char endpoint, unsigned char *data, int length, int *transferred, unsigned int timeout)

  Perform a USB interrupt transfer.

34.268 lib/ndkbuild/libusb/libusb/version.h File Reference

```
#include "version_nano.h"
```

Macros

- #define LIBUSB_MINOR 0
- #define LIBUSB_MICRO 21
- #define LIBUSB_RC "-rc3"

34.268.1 Macro Definition Documentation

#define LIBUSB_MINOR 0

#define LIBUSB_MICRO 21

#define LIBUSB_RC "-rc3"

34.269 lib/ndkbuild/libusb/libusb/version_nano.h File Reference

Macros

- #define LIBUSB_NANO 11140

34.269.1 Macro Definition Documentation

#define LIBUSB_NANO 11140

34.270 lib/ndkbuild/libusb/msvc/errno.h File Reference

```
#include <crtdefs.h>
```

Macros

- #define EPERM 1 /∗ Operation not permitted ∗/
- #define ENOFILE 2 /∗ No such file or directory ∗/
- #define ENOENT 2
- #define ESRCH 3 /∗ No such process ∗/
- #define EINTR 4 /∗ Interrupted function call ∗/
- #define EIO 5 /∗ Input/output error ∗/
- #define ENXIO 6 /∗ No such device or address ∗/
- #define E2BIG 7 /∗ Arg list too long ∗/
- #define ENOEXEC 8 /∗ Exec format error ∗/
- #define EBADF 9 /∗ Bad file descriptor ∗/
- #define ECHILD 10 /∗ No child processes ∗/
- #define EAGAIN 11 /∗ Resource temporarily unavailable ∗/
- #define ENOMEM 12 /∗ Not enough space ∗/
- #define EACCES 13 /∗ Permission denied ∗/
- #define EFAULT 14 /∗ Bad address ∗/
- #define EBUSY 16 /∗ strerror reports "Resource device" ∗/
- #define EEXIST 17 /∗ File exists ∗/
- #define EXDEV 18 /∗ Improper link (cross-device link?) ∗/
- #define ENODEV 19 /∗ No such device ∗/
- #define ENOTDIR 20 /∗ Not a directory ∗/
- #define EISDIR 21 /∗ Is a directory ∗/
- #define EINVAL 22 /∗ Invalid argument ∗/
- #define ENFILE 23 /∗ Too many open files in system ∗/
- #define EMFILE 24 /∗ Too many open files ∗/
- #define ENOTTY 25 /∗ Inappropriate I/O control operation ∗/
- #define EFBIG 27 /∗ File too large ∗/
- #define ENOSPC 28 /∗ No space left on device ∗/
- #define ESPIPE 29 /∗ Invalid seek (seek on a pipe?) ∗/
- #define EROFS 30 /∗ Read-only file system ∗/
- #define EMLINK 31 /∗ Too many links ∗/
- #define EPIPE 32 /∗ Broken pipe ∗/
- #define EDOM 33 /∗ Domain error (math functions) ∗/
- #define ERANGE 34 /∗ Result too large (possibly too small) ∗/
- #define EDEADLOCK 36 /∗ Resource deadlock avoided (non-Cyg) ∗/
- #define EDEADLK 36
- #define errno (∗_errno())

34.270.1 Macro Definition Documentation

#define EPERM 1 /∗ Operation not permitted ∗/

#define ENOFILE 2 /∗ No such file or directory ∗/

#define ENOENT 2

#define ESRCH 3 /∗ No such process ∗/

#define EINTR 4 /∗ Interrupted function call ∗/

#define EIO 5 /∗ Input/output error ∗/

#define ENXIO 6 /∗ No such device or address ∗/

#define E2BIG 7 /∗ Arg list too long ∗/

#define ENOEXEC 8 /∗ Exec format error ∗/

#define EBADF 9 /∗ Bad file descriptor ∗/

#define ECHILD 10 /∗ No child processes ∗/

#define EAGAIN 11 /∗ Resource temporarily unavailable ∗/

#define ENOMEM 12 /∗ Not enough space ∗/

#define EACCES 13 /∗ Permission denied ∗/

#define EFAULT 14 /∗ Bad address ∗/

#define EBUSY 16 /∗ strerror reports "Resource device" ∗/

#define EEXIST 17 /∗ File exists ∗/

#define EXDEV 18 /∗ Improper link (cross-device link?) ∗/

#define ENODEV 19 /∗ No such device ∗/

#define ENOTDIR 20 /∗ Not a directory ∗/

#define EISDIR 21 /∗ Is a directory ∗/

#define EINVAL 22 /∗ Invalid argument ∗/

#define ENFILE 23 /∗ Too many open files in system ∗/

#define EMFILE 24 /∗ Too many open files ∗/

#define ENOTTY 25 /∗ Inappropriate I/O control operation ∗/

#define EFBIG 27 /∗ File too large ∗/

#define ENOSPC 28 /∗ No space left on device ∗/

#define ESPIPE 29 /∗ Invalid seek (seek on a pipe?) ∗/

#define EROFS 30 /∗ Read-only file system ∗/

#define EMLINK 31 /∗ Too many links ∗/

#define EPIPE 32 /∗ Broken pipe ∗/

#define EDOM 33 /∗ Domain error (math functions) ∗/

#define ERANGE 34 /∗ Result too large (possibly too small) ∗/

#define EDEADLOCK 36 /∗ Resource deadlock avoided (non-Cyg) ∗/

#define EDEADLK 36

#define errno (∗_errno())

34.271 lib/ndkbuild/libusb/msvc/inttypes.h File Reference

```
#include <stdint.h>
```

Data Structures

- struct imaxdiv_t

  This file has no copyright assigned and is placed in the Public Domain.

Macros

- #define PRId8 "d"
- #define PRId16 "d"
- #define PRId32 "d"
- #define PRId64 "I64d"

- #define PRIdLEAST8 "d"
- #define PRIdLEAST16 "d"
- #define PRIdLEAST32 "d"
- #define PRIdLEAST64 "l64d"
- #define PRIdFAST8 "d"
- #define PRIdFAST16 "d"
- #define PRIdFAST32 "d"
- #define PRIdFAST64 "l64d"
- #define PRIdMAX "l64d"
- #define PRIi8 "i"
- #define PRIi16 "i"
- #define PRIi32 "i"
- #define PRIi64 "l64i"
- #define PRIiLEAST8 "i"
- #define PRIiLEAST16 "i"
- #define PRIiLEAST32 "i"
- #define PRIiLEAST64 "l64i"
- #define PRIiFAST8 "i"
- #define PRIiFAST16 "i"
- #define PRIiFAST32 "i"
- #define PRIiFAST64 "l64i"
- #define PRIiMAX "l64i"
- #define PRIo8 "o"
- #define PRIo16 "o"
- #define PRIo32 "o"
- #define PRIo64 "l64o"
- #define PRIoLEAST8 "o"
- #define PRIoLEAST16 "o"
- #define PRIoLEAST32 "o"
- #define PRIoLEAST64 "l64o"
- #define PRIoFAST8 "o"
- #define PRIoFAST16 "o"
- #define PRIoFAST32 "o"
- #define PRIoFAST64 "l64o"
- #define PRIoMAX "l64o"
- #define PRIu8 "u"
- #define PRIu16 "u"
- #define PRIu32 "u"
- #define PRIu64 "l64u"
- #define PRIuLEAST8 "u"
- #define PRIuLEAST16 "u"
- #define PRIuLEAST32 "u"
- #define PRIuLEAST64 "l64u"
- #define PRIuFAST8 "u"
- #define PRIuFAST16 "u"
- #define PRIuFAST32 "u"
- #define PRIuFAST64 "l64u"
- #define PRIuMAX "l64u"
- #define PRIx8 "x"
- #define PRIx16 "x"
- #define PRIx32 "x"
- #define PRIx64 "l64x"
- #define PRIxLEAST8 "x"
- #define PRIxLEAST16 "x"
- #define PRIxLEAST32 "x"
- #define PRIxLEAST64 "l64x"
- #define PRIxFAST8 "x"
- #define PRIxFAST16 "x"
- #define PRIxFAST32 "x"
- #define PRIxFAST64 "l64x"
- #define PRIxMAX "l64x"
- #define PRIX8 "X"
- #define PRIX16 "X"
- #define PRIX32 "X"
- #define PRIX64 "l64X"
- #define PRIXLEAST8 "X"
- #define PRIXLEAST16 "X"
- #define PRIXLEAST32 "X"

- #define PRIXLEAST64 "I64X"
- #define PRIXFAST8 "X"
- #define PRIXFAST16 "X"
- #define PRIXFAST32 "X"
- #define PRIXFAST64 "I64X"
- #define PRIXMAX "I64X"
- #define SCNd16 "hd"
- #define SCNd32 "d"
- #define SCNd64 "I64d"
- #define SCNdLEAST16 "hd"
- #define SCNdLEAST32 "d"
- #define SCNdLEAST64 "I64d"
- #define SCNdFAST16 "hd"
- #define SCNdFAST32 "d"
- #define SCNdFAST64 "I64d"
- #define SCNdMAX "I64d"
- #define SCNi16 "hi"
- #define SCNi32 "i"
- #define SCNi64 "I64i"
- #define SCNiLEAST16 "hi"
- #define SCNiLEAST32 "i"
- #define SCNiLEAST64 "I64i"
- #define SCNiFAST16 "hi"
- #define SCNiFAST32 "i"
- #define SCNiFAST64 "I64i"
- #define SCNiMAX "I64i"
- #define SCNo16 "ho"
- #define SCNo32 "o"
- #define SCNo64 "I64o"
- #define SCNoLEAST16 "ho"
- #define SCNoLEAST32 "o"
- #define SCNoLEAST64 "I64o"
- #define SCNoFAST16 "ho"
- #define SCNoFAST32 "o"
- #define SCNoFAST64 "I64o"
- #define SCNoMAX "I64o"
- #define SCNx16 "hx"
- #define SCNx32 "x"
- #define SCNx64 "I64x"
- #define SCNxLEAST16 "hx"
- #define SCNxLEAST32 "x"
- #define SCNxLEAST64 "I64x"
- #define SCNxFAST16 "hx"
- #define SCNxFAST32 "x"
- #define SCNxFAST64 "I64x"
- #define SCNxMAX "I64x"
- #define SCNu16 "hu"
- #define SCNu32 "u"
- #define SCNu64 "I64u"
- #define SCNuLEAST16 "hu"
- #define SCNuLEAST32 "u"
- #define SCNuLEAST64 "I64u"
- #define SCNuFAST16 "hu"
- #define SCNuFAST32 "u"
- #define SCNuFAST64 "I64u"
- #define SCNuMAX "I64u"
- #define PRIdPTR "d"
- #define PRIiPTR "i"
- #define PRIoPTR "o"
- #define PRIuPTR "u"
- #define PRIxPTR "x"
- #define PRIXPTR "X"
- #define SCNdPTR "d"
- #define SCNiPTR "i"
- #define SCNoPTR "o"
- #define SCNxPTR "x"
- #define SCNuPTR "u"

34.271.1 Macro Definition Documentation

#define PRId8 "d"

#define PRId16 "d"

#define PRId32 "d"

#define PRId64 "I64d"

#define PRIdLEAST8 "d"

#define PRIdLEAST16 "d"

#define PRIdLEAST32 "d"

#define PRIdLEAST64 "I64d"

#define PRIdFAST8 "d"

#define PRIdFAST16 "d"

#define PRIdFAST32 "d"

#define PRIdFAST64 "I64d"

#define PRIdMAX "I64d"

#define PRIi8 "i"

#define PRIi16 "i"

#define PRIi32 "i"

#define PRIi64 "I64i"

#define PRIiLEAST8 "i"

#define PRIiLEAST16 "i"

#define PRIiLEAST32 "i"

#define PRIiLEAST64 "I64i"

#define PRIiFAST8 "i"

#define PRIiFAST16 "i"

#define PRIiFAST32 "i"

#define PRIiFAST64 "I64i"

#define PRIiMAX "I64i"

#define PRIo8 "o"

#define PRIo16 "o"

#define PRIo32 "o"

#define PRIo64 "I64o"

#define PRIoLEAST8 "o"

#define PRIoLEAST16 "o"

#define PRIoLEAST32 "o"

#define PRIoLEAST64 "I64o"

```
#define PRIoFAST8 "o"

#define PRIoFAST16 "o"

#define PRIoFAST32 "o"

#define PRIoFAST64 "I64o"

#define PRIoMAX "I64o"

#define PRIu8 "u"

#define PRIu16 "u"

#define PRIu32 "u"

#define PRIu64 "I64u"

#define PRIuLEAST8 "u"

#define PRIuLEAST16 "u"

#define PRIuLEAST32 "u"

#define PRIuLEAST64 "I64u"

#define PRIuFAST8 "u"

#define PRIuFAST16 "u"

#define PRIuFAST32 "u"

#define PRIuFAST64 "I64u"

#define PRIuMAX "I64u"

#define PRIx8 "x"

#define PRIx16 "x"

#define PRIx32 "x"

#define PRIx64 "I64x"

#define PRIxLEAST8 "x"

#define PRIxLEAST16 "x"

#define PRIxLEAST32 "x"

#define PRIxLEAST64 "I64x"

#define PRIxFAST8 "x"

#define PRIxFAST16 "x"

#define PRIxFAST32 "x"

#define PRIxFAST64 "I64x"

#define PRIxMAX "I64x"

#define PRIX8 "X"

#define PRIX16 "X"

#define PRIX32 "X"

#define PRIX64 "I64X"
```

```
#define PRIXLEAST8 "X"

#define PRIXLEAST16 "X"

#define PRIXLEAST32 "X"

#define PRIXLEAST64 "I64X"

#define PRIXFAST8 "X"

#define PRIXFAST16 "X"

#define PRIXFAST32 "X"

#define PRIXFAST64 "I64X"

#define PRIXMAX "I64X"

#define SCNd16 "hd"

#define SCNd32 "d"

#define SCNd64 "I64d"

#define SCNdLEAST16 "hd"

#define SCNdLEAST32 "d"

#define SCNdLEAST64 "I64d"

#define SCNdFAST16 "hd"

#define SCNdFAST32 "d"

#define SCNdFAST64 "I64d"

#define SCNdMAX "I64d"

#define SCNi16 "hi"

#define SCNi32 "i"

#define SCNi64 "I64i"

#define SCNiLEAST16 "hi"

#define SCNiLEAST32 "i"

#define SCNiLEAST64 "I64i"

#define SCNiFAST16 "hi"

#define SCNiFAST32 "i"

#define SCNiFAST64 "I64i"

#define SCNiMAX "I64i"

#define SCNo16 "ho"

#define SCNo32 "o"

#define SCNo64 "I64o"

#define SCNoLEAST16 "ho"

#define SCNoLEAST32 "o"

#define SCNoLEAST64 "I64o"
```

```
#define SCNoFAST16 "ho"

#define SCNoFAST32 "o"

#define SCNoFAST64 "I64o"

#define SCNoMAX "I64o"

#define SCNx16 "hx"

#define SCNx32 "x"

#define SCNx64 "I64x"

#define SCNxLEAST16 "hx"

#define SCNxLEAST32 "x"

#define SCNxLEAST64 "I64x"

#define SCNxFAST16 "hx"

#define SCNxFAST32 "x"

#define SCNxFAST64 "I64x"

#define SCNxMAX "I64x"

#define SCNu16 "hu"

#define SCNu32 "u"

#define SCNu64 "I64u"

#define SCNuLEAST16 "hu"

#define SCNuLEAST32 "u"

#define SCNuLEAST64 "I64u"

#define SCNuFAST16 "hu"

#define SCNuFAST32 "u"

#define SCNuFAST64 "I64u"

#define SCNuMAX "I64u"

#define PRIdPTR "d"

#define PRIiPTR "i"

#define PRIoPTR "o"

#define PRIuPTR "u"

#define PRIxPTR "x"

#define PRIXPTR "X"

#define SCNdPTR "d"

#define SCNiPTR "i"

#define SCNoPTR "o"

#define SCNxPTR "x"

#define SCNuPTR "u"
```

34.272 lib/ndkbuild/libusb/msvc/missing.c File Reference

```
#include "missing.h"
#include <config.h>
#include <libusbi.h>
#include <windows.h>
```

Macros

- #define ENVIRONMENT_REG_PATH _T("Software\\libusb\\environment")

Functions

- char * getenv (const char *name)

34.272.1 Macro Definition Documentation

#define ENVIRONMENT_REG_PATH _T("Software\\libusb\\environment")

34.272.2 Function Documentation

char* getenv (const char * *name*)

34.273 lib/ndkbuild/libusb/msvc/missing.h File Reference

Macros

- #define SleepEx(m, a) Sleep(m)

Functions

- char * getenv (const char *name)

34.273.1 Macro Definition Documentation

#define SleepEx(*m*, *a*) Sleep(m)

34.273.2 Function Documentation

char* getenv (const char * *name*)

34.274 lib/ndkbuild/libusb/msvc/stdint.h File Reference

Macros

- #define INT8_MIN (-128)
- #define INT16_MIN (-32768)
- #define INT32_MIN (-2147483647 - 1)
- #define INT64_MIN (-9223372036854775807LL - 1)
- #define INT8_MAX 127
- #define INT16_MAX 32767
- #define INT32_MAX 2147483647
- #define INT64_MAX 9223372036854775807LL
- #define UINT8_MAX 255
- #define UINT16_MAX 65535
- #define UINT32_MAX 0xffffffffU /* 4294967295U */
- #define UINT64_MAX 0xffffffffffffffffULL /* 18446744073709551615ULL */
- #define INT_LEAST8_MIN INT8_MIN
- #define INT_LEAST16_MIN INT16_MIN
- #define INT_LEAST32_MIN INT32_MIN
- #define INT_LEAST64_MIN INT64_MIN
- #define INT_LEAST8_MAX INT8_MAX
- #define INT_LEAST16_MAX INT16_MAX

- #define INT_LEAST32_MAX INT32_MAX
- #define INT_LEAST64_MAX INT64_MAX
- #define UINT_LEAST8_MAX UINT8_MAX
- #define UINT_LEAST16_MAX UINT16_MAX
- #define UINT_LEAST32_MAX UINT32_MAX
- #define UINT_LEAST64_MAX UINT64_MAX
- #define INT_FAST8_MIN INT8_MIN
- #define INT_FAST16_MIN INT16_MIN
- #define INT_FAST32_MIN INT32_MIN
- #define INT_FAST64_MIN INT64_MIN
- #define INT_FAST8_MAX INT8_MAX
- #define INT_FAST16_MAX INT16_MAX
- #define INT_FAST32_MAX INT32_MAX
- #define INT_FAST64_MAX INT64_MAX
- #define UINT_FAST8_MAX UINT8_MAX
- #define UINT_FAST16_MAX UINT16_MAX
- #define UINT_FAST32_MAX UINT32_MAX
- #define UINT_FAST64_MAX UINT64_MAX
- #define INTPTR_MIN INT32_MIN
- #define INTPTR_MAX INT32_MAX
- #define UINTPTR_MAX UINT32_MAX
- #define INTMAX_MIN INT64_MIN
- #define INTMAX_MAX INT64_MAX
- #define UINTMAX_MAX UINT64_MAX
- #define PTRDIFF_MIN INT32_MIN
- #define PTRDIFF_MAX INT32_MAX
- #define SIG_ATOMIC_MIN INT32_MIN
- #define SIG_ATOMIC_MAX INT32_MAX
- #define SIZE_MAX UINT32_MAX
- #define WCHAR_MIN 0U
- #define WCHAR_MAX 0xffffU
- #define WINT_MIN 0U
- #define WINT_MAX 0xffffU
- #define INT8_C(val) (INT_LEAST8_MAX-INT_LEAST8_MAX+(val))
- #define INT16_C(val) (INT_LEAST16_MAX-INT_LEAST16_MAX+(val))
- #define INT32_C(val) (INT_LEAST32_MAX-INT_LEAST32_MAX+(val))
- #define INT64_C(val) val##i64
- #define UINT8_C(val) (val)
- #define UINT16_C(val) (val)
- #define UINT32_C(val) (val##i32)
- #define UINT64_C(val) val##ui64
- #define INTMAX_C(val) val##i64
- #define UINTMAX_C(val) val##ui64

Typedefs

- typedef int intptr_t
- typedef unsigned int uintptr_t
- typedef int ptrdiff_t
- typedef unsigned short wchar_t
- typedef unsigned short wint_t
- typedef unsigned short wctype_t
- typedef __int8 int8_t
- typedef unsigned __int8 uint8_t
- typedef __int16 int16_t
- typedef unsigned __int16 uint16_t
- typedef __int32 int32_t
- typedef unsigned __int32 uint32_t
- typedef __int64 int64_t
- typedef unsigned __int64 uint64_t
- typedef signed char int_least8_t
- typedef unsigned char uint_least8_t
- typedef short int_least16_t
- typedef unsigned short uint_least16_t
- typedef int int_least32_t
- typedef unsigned uint_least32_t
- typedef __int64 int_least64_t

- typedef unsigned __int64 uint_least64_t
- typedef __int8 int_fast8_t
- typedef unsigned __int8 uint_fast8_t
- typedef __int16 int_fast16_t
- typedef unsigned __int16 uint_fast16_t
- typedef __int32 int_fast32_t
- typedef unsigned __int32 uint_fast32_t
- typedef __int64 int_fast64_t
- typedef unsigned __int64 uint_fast64_t
- typedef __int64 intmax_t
- typedef unsigned __int64 uintmax_t

34.274.1 Macro Definition Documentation

#define INT8_MIN (-128)

#define INT16_MIN (-32768)

#define INT32_MIN (-2147483647 - 1)

#define INT64_MIN (-9223372036854775807LL - 1)

#define INT8_MAX 127

#define INT16_MAX 32767

#define INT32_MAX 2147483647

#define INT64_MAX 9223372036854775807LL

#define UINT8_MAX 255

#define UINT16_MAX 65535

#define UINT32_MAX 0xffffffffU /∗ 4294967295U ∗/

#define UINT64_MAX 0xffffffffffffffffULL /∗ 18446744073709551615ULL ∗/

#define INT_LEAST8_MIN INT8_MIN

#define INT_LEAST16_MIN INT16_MIN

#define INT_LEAST32_MIN INT32_MIN

#define INT_LEAST64_MIN INT64_MIN

#define INT_LEAST8_MAX INT8_MAX

#define INT_LEAST16_MAX INT16_MAX

#define INT_LEAST32_MAX INT32_MAX

#define INT_LEAST64_MAX INT64_MAX

#define UINT_LEAST8_MAX UINT8_MAX

#define UINT_LEAST16_MAX UINT16_MAX

#define UINT_LEAST32_MAX UINT32_MAX

#define UINT_LEAST64_MAX UINT64_MAX

#define INT_FAST8_MIN INT8_MIN

#define INT_FAST16_MIN INT16_MIN

#define INT_FAST32_MIN INT32_MIN

#define INT_FAST64_MIN **INT64_MIN**

#define INT_FAST8_MAX **INT8_MAX**

#define INT_FAST16_MAX **INT16_MAX**

#define INT_FAST32_MAX **INT32_MAX**

#define INT_FAST64_MAX **INT64_MAX**

#define UINT_FAST8_MAX **UINT8_MAX**

#define UINT_FAST16_MAX **UINT16_MAX**

#define UINT_FAST32_MAX **UINT32_MAX**

#define UINT_FAST64_MAX **UINT64_MAX**

#define INTPTR_MIN **INT32_MIN**

#define INTPTR_MAX **INT32_MAX**

#define UINTPTR_MAX **UINT32_MAX**

#define INTMAX_MIN **INT64_MIN**

#define INTMAX_MAX **INT64_MAX**

#define UINTMAX_MAX **UINT64_MAX**

#define PTRDIFF_MIN **INT32_MIN**

#define PTRDIFF_MAX **INT32_MAX**

#define SIG_ATOMIC_MIN **INT32_MIN**

#define SIG_ATOMIC_MAX **INT32_MAX**

#define SIZE_MAX **UINT32_MAX**

#define WCHAR_MIN 0U

#define WCHAR_MAX 0xffffU

#define WINT_MIN 0U

#define WINT_MAX 0xffffU

#define INT8_C(*val*) (**INT_LEAST8_MAX-INT_LEAST8_MAX**+(val))

#define INT16_C(*val*) (**INT_LEAST16_MAX-INT_LEAST16_MAX**+(val))

#define INT32_C(*val*) (**INT_LEAST32_MAX-INT_LEAST32_MAX**+(val))

#define INT64_C(*val*) val##i64

#define UINT8_C(*val*) (val)

#define UINT16_C(*val*) (val)

#define UINT32_C(*val*) (val##i32)

#define UINT64_C(*val*) val##ui64

#define INTMAX_C(*val*) val##i64

#define UINTMAX_C(*val*) val##ui64

34.274.2 Typedef Documentation

typedef int **intptr_t**

typedef unsigned int **uintptr_t**

typedef int **ptrdiff_t**

typedef unsigned short **wchar_t**

typedef unsigned short **wint_t**

typedef unsigned short **wctype_t**

typedef __int8 **int8_t**

typedef unsigned __int8 **uint8_t**

typedef __int16 **int16_t**

typedef unsigned __int16 **uint16_t**

typedef __int32 **int32_t**

typedef unsigned __int32 **uint32_t**

typedef __int64 **int64_t**

typedef unsigned __int64 **uint64_t**

typedef signed char **int_least8_t**

typedef unsigned char **uint_least8_t**

typedef short **int_least16_t**

typedef unsigned short **uint_least16_t**

typedef int **int_least32_t**

typedef unsigned **uint_least32_t**

typedef __int64 **int_least64_t**

typedef unsigned __int64 **uint_least64_t**

typedef __int8 **int_fast8_t**

typedef unsigned __int8 **uint_fast8_t**

typedef __int16 **int_fast16_t**

typedef unsigned __int16 **uint_fast16_t**

typedef __int32 **int_fast32_t**

typedef unsigned __int32 **uint_fast32_t**

typedef __int64 **int_fast64_t**

typedef unsigned __int64 **uint_fast64_t**

typedef __int64 **intmax_t**

typedef unsigned __int64 **uintmax_t**

34.275 lib/ndkbuild/libusb/tests/libusb_testlib.h File Reference

```
#include <stdio.h>
```

Data Structures

- struct libusb_testlib_ctx

  Context for test library functions.
- struct libusb_testlib_test

  Structure holding a test description.

Macros

- #define bool int
- #define true (1 == 1)
- #define false (!true)
- #define LIBUSB_NULL_TEST {NULL, NULL}

  Value to use at the end of a test array to indicate the last element.

Typedefs

- typedef libusb_testlib_result(∗ libusb_testlib_test_function)(libusb_testlib_ctx ∗ctx)

  Function pointer for a libusb test function.

Enumerations

- enum libusb_testlib_result { TEST_STATUS_SUCCESS, TEST_STATUS_FAILURE, TEST_STATUS_ERROR, TEST_STATUS_SKIP }

  Values returned from a test function to indicate test result.

Functions

- void libusb_testlib_logf (libusb_testlib_ctx ∗ctx, const char ∗fmt,...)

  Logs some test information or state.
- int libusb_testlib_run_tests (int argc, char ∗∗argv, const libusb_testlib_test ∗tests)

  Runs the tests provided.

34.275.1 Macro Definition Documentation

#define bool int

#define true (1 == 1)

#define false (!true)

#define LIBUSB_NULL_TEST {NULL, NULL}

Value to use at the end of a test array to indicate the last element.

34.275.2 Typedef Documentation

typedef libusb_testlib_result(∗ libusb_testlib_test_function)(libusb_testlib_ctx ∗ctx)

Function pointer for a libusb test function.

Should return TEST_STATUS_SUCCESS on success or another TEST_STATUS value.

34.275.3 Enumeration Type Documentation

enum libusb_testlib_result

Values returned from a test function to indicate test result.

Enumerator

TEST_STATUS_SUCCESS Indicates that the test ran successfully.

TEST_STATUS_FAILURE Indicates that the test failed one or more test.

TEST_STATUS_ERROR Indicates that an unexpected error occurred.

TEST_STATUS_SKIP Indicates that the test can't be run. For example this may be due to no suitable device being connected to perform the tests.

34.275.4 Function Documentation

void libusb_testlib_logf (libusb_testlib_ctx * *ctx*, const char * *fmt*, ...)

Logs some test information or state.

int libusb_testlib_run_tests (int *argc*, char ** *argv*, const libusb_testlib_test * *tests*)

Runs the tests provided.

Before running any tests argc and argv will be processed to determine the mode of operation.

Parameters

argc	The argc from main
argv	The argv from main
tests	A NULL_TEST terminated array of tests

Returns

0 on success, non-zero on failure

34.276 lib/ndkbuild/libusb/tests/stress.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include "libusb.h"
#include "libusb_testlib.h"
```

Macros

- #define LIST_COUNT 100

Functions

- int main (int argc, char **argv)

34.276.1 Macro Definition Documentation

#define LIST_COUNT 100

34.276.2 Function Documentation

int main (int *argc*, char ** *argv*)

34.277 lib/ndkbuild/libusb/tests/testlib.c File Reference

```
#include "libusb_testlib.h"
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

Macros

- #define NULL_PATH "/dev/null"
- #define INVALID_FD -1
- #define IGNORE_RETVAL(expr) do { (void)(expr); } while(0)

Functions

- void libusb_testlib_logf (libusb_testlib_ctx *ctx, const char *fmt,...)

  Logs some test information or state.
- int libusb_testlib_run_tests (int argc, char **argv, const libusb_testlib_test *tests)

  Runs the tests provided.

34.277.1 Macro Definition Documentation

#define NULL_PATH "/dev/null"

#define INVALID_FD -1

#define IGNORE_RETVAL(*expr*) do { (void)(expr); } while(0)

34.277.2 Function Documentation

void libusb_testlib_logf (libusb_testlib_ctx * *ctx,* const char * *fmt,* ...)

Logs some test information or state.

int libusb_testlib_run_tests (int *argc,* char ** *argv,* const libusb_testlib_test * *tests*)

Runs the tests provided.

Before running any tests argc and argv will be processed to determine the mode of operation.

Parameters

argc	The argc from main
argv	The argv from main
tests	A NULL_TEST terminated array of tests

Returns

0 on success, non-zero on failure

34.278 lib/wrappers/scilab/foo.c File Reference

```
#include "foo.h"
#include <stdio.h>
```

Functions

- void foo_something ()

34.278.1 Function Documentation

void foo_something ( )

34.279 lib/wrappers/scilab/foo.h File Reference

Functions

- void foo_something ()

34.279.1 Function Documentation

void foo_something ( )

34.280 samples/TestLib/aiocommon.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <ctype.h>
#include "aiocommon.h"
#include "aiousb.h"
```

Macros

- #define DUMP 0x1000
- #define CNTS 0x1001
- #define JCONF 0x1002
- #define REPEAT 0x1003

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_aio_cmd_line (struct opts ∗options, int argc, char ∗argv[])

  Simple command line parser sets up testing features.
- void print_aio_usage (int argc, char ∗∗argv, struct option ∗options)
- AIORET_TYPE aio_list_devices (struct opts ∗options, int ∗indices, int num_devices)
- AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗config, struct opts ∗options)
- AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗options)
- AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗buf, struct opts ∗options)

Variables

- struct opts AIO_OPTIONS

34.280.1 Macro Definition Documentation

#define DUMP 0x1000

#define CNTS 0x1001

#define JCONF 0x1002

#define REPEAT 0x1003

34.280.2 Function Documentation

struct channel_range∗ get_channel_range (char ∗ *optarg*)

void process_aio_cmd_line (struct opts ∗ *options,* int *argc,* char ∗ *argv[]*)

Simple command line parser sets up testing features.

void print_aio_usage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

AIORET_TYPE aio_list_devices (struct opts ∗ *options,* int ∗ *indices,* int *num_devices*)

AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗ *config,* struct opts ∗ *options*)

AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗ *options*)

AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗ *buf,* struct opts ∗ *options*)

34.280.3 Variable Documentation

struct opts AIO_OPTIONS

34.281 samples/TestLib/aiocommon.h File Reference

```
#include "aiousb.h"
#include <getopt.h>
#include <stdint.h>
```

Data Structures

- struct channel_range
- struct opts

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_aio_cmd_line (struct opts ∗options, int argc, char ∗argv[])

  Simple command line parser sets up testing features.
- void print_aio_usage (int argc, char ∗∗argv, struct option ∗options)
- AIORET_TYPE aio_list_devices (struct opts ∗options, int ∗indices, int num_devices)
- AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗buf, struct opts ∗options)
- AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗config, struct opts ∗options)
- AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗options)

Variables

- struct opts AIO_OPTIONS

34.281.1 Function Documentation

struct channel_range∗ get_channel_range (char ∗ *optarg*)

void process_aio_cmd_line (struct opts ∗ *options,* int *argc,* char ∗ *argv[]*)

Simple command line parser sets up testing features.

void print_aio_usage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

AIORET_TYPE aio_list_devices (struct opts ∗ *options,* int ∗ *indices,* int *num_devices*)

AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗ *buf,* struct opts ∗ *options*)

AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗ *config,* struct opts ∗ *options*)

AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗ *options*)

34.281.2 Variable Documentation

struct opts AIO_OPTIONS

34.282 samples/TestLib/TestCaseSetup.cpp File Reference

```
#include "TestCaseSetup.h"
#include <stdlib.h>
#include <aiousb.h>
#include <AIOUSB_Core.h>
#include "AIOTypes.h"
```

Variables

- int CURRENT_DEBUG_LEVEL = 1

34.282.1 Variable Documentation

int CURRENT_DEBUG_LEVEL = 1

34.283 samples/TestLib/TestCaseSetup.h File Reference

```
#include <aiousb.h>
#include <exception>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <unistd.h>
#include <stdarg.h>
#include "AIOUSB_Core.h"
```

Data Structures

- class Error
- class TestCaseSetup

Macros

- #define ERROR_LEVEL 2<<1
- #define FATAL_LEVEL 2<<1
- #define ALERT_LEVEL 2<<2
- #define WARN_LEVEL 2<<3
- #define INFO_LEVEL 2<<4
- #define DEBUG_LEVEL 2<<5
- #define TRACE_LEVEL 2<<6
- #define LOG(X,...) printf(X, ##__VA_ARGS__);
- #define INFO(X,...)
- #define TRACE(X,...)
- #define DEBUG(X,...)
- #define ERROR(X,...)
- #define FATAL(X,...)
- #define TERSE_LOGGING (WARN_LEVEL | ERROR_LEVEL | INFO_LEVEL)
- #define VERBOSE_LOGGING (DEBUG_LEVEL | INFO_LEVEL | WARN_LEVEL | ERROR_LEVEL)
- #define THROW_ERROR(x) ThrowError(x , __LINE__)
- #define CHECK_RESULT(x) if(result != AIOUSB_SUCCESS) ThrowError(result,__LINE__);

Variables

- const int MAX_NAME_SIZE = 20
- const int DEF_MAX_CHANNELS = 128
- const int DEF_NUM_CHANNELS = 16
- const int DEF_CAL_CHANNEL = 5
- int CURRENT_DEBUG_LEVEL

34.283.1 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__);

34.283.2 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

34.284 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)

34.284.1 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*)

34.285 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)

34.285.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.285.2 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice * *dev*)

34.286 samples/USB-AI16-16/continuous_mode.c File Reference

```
#include <aiousb.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL fnd (AIOUSBDevice *dev)

34.286.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.286.2 Function Documentation

AIOUSB_BOOL fnd (AIOUSBDevice * *dev*)

34.287 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

34.287.1 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[]*)

34.287.2 Variable Documentation

FILE∗ fp

34.288 samples/USB-AI16-16/continuous_mode_from_json_config.c File Reference

```
#include "aiocommon.h"
#include <aiousb.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

34.288.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.288.2 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[]*)

34.288.3 Variable Documentation

FILE* fp

34.289 samples/USB-AI16-16/daitest.cpp File Reference

```
#include <iostream>
#include <thread>
#include <chrono>
#include "aiousb.h"
#include <signal.h>
#include <unistd.h>
```

Functions

- void handle_signal (int signal)
- int main (int argc, char *argv[])

Variables

- int exit_sample = 0
- struct sigaction old_action

34.289.1 Function Documentation

void handle_signal (int *signal*)

int main (int *argc,* char * *argv[]*)

34.289.2 Variable Documentation

int exit_sample = 0

struct sigaction old_action

34.290 samples/USB-AI16-16/dio_sample.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice *dev)
- int main (int argc, char **argv)

34.290.1 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

34.291 samples/USB-AI16-16/diotest.c File Reference

```
#include <stdio.h>
#include "aiousb.h"
```

Functions

- int main (int argc, char ∗argv[])

34.291.1 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

34.292 samples/USB-AI16-16/diotest2.cpp File Reference

```
#include <iostream>
#include "aiousb.h"
```

Functions

- int main (int argc, char ∗argv[])

34.292.1 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

34.293 samples/USB-AI16-16/HOLD/dirktest.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include <AIODataTypes.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

34.293.1 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[]*)

34.293.2 Variable Documentation

FILE∗ fp

34.294 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)

34.294.1 Function Documentation

AIOUSB_BOOL find_ai_board (**AIOUSBDevice** ∗ *dev*)

int main (int *argc,* char ∗∗ *argv*)

34.295 samples/USB-AI16-16/HOLD/reverse_cal_table.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- void goDoIt (TestCaseSetup &t)
- int main (int argc, char ∗∗argv)

34.295.1 Function Documentation

void goDoIt (**TestCaseSetup** & *t*)

int main (int *argc,* char ∗∗ *argv*)

34.296 samples/USB-AI16-16/reverse_cal_table.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- void goDoIt (TestCaseSetup &t)
- int main (int argc, char ∗∗argv)

34.296.1 Function Documentation

void goDoIt (TestCaseSetup & *t*)

int main (int *argc,* char ** *argv*)

34.297 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)

34.297.1 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

34.298 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)

34.298.1 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*)

34.299 samples/USB-AI16-16/HOLD/test.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include <AIODataTypes.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

34.299.1 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[]*)

34.299.2 Variable Documentation

FILE∗ fp

34.300 samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp File Reference

```
#include <iostream>
#include <iterator>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <aiousb.h>
#include <AIOUSB_Core.h>
#include <USBDeviceManager.hpp>
#include <USB_AI16_Family.hpp>
```

Functions

- int main (int argc, char ∗argv[])

34.300.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Author

Jimi Damon jdamon@accesio.com

Version

Format:

h

34.300.2 Function Documentation

int main (int *argc*, char * *argv[]*)

34.301 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)

34.301.1 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

34.302 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)

34.302.1 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

34.303 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)

34.303.1 Function Documentation

struct options get_options (int *argc,* char ** *argv*)

int main (int *argc,* char ** *argv*)

34.304 samples/USB-AI16-16/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
```

Functions

- int main (int argc, char **argv)

34.304.1 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

34.305 samples/USB-AO16-16/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
```

Functions

- int main (int argc, char **argv)

34.305.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.306 samples/USB-DA12-8A/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
```

Functions

- int main (int argc, char **argv)

34.306.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.307 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)

34.307.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.308 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)

34.308.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.309 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

34.309.1 Detailed Description

Sample that demonstrates data ac.

Author

Jimi Damon `james.damon@accesio.com`

Date

Thu Nov 12 10:54:48 2015

34.309.2 Function Documentation

struct channel_range* get_channel_range (char * *optarg*)

void process_cmd_line (struct opts * , int *argc,* char * *argv[]*)

AIOUSB_BOOL fnd (AIOUSBDevice * *dev*)

int main (int *argc,* char * *argv[]*)

Start with the NewAIOContinousBufFromJSON("{'aiocontin

34.309.3 Variable Documentation

FILE* fp

34.310 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

34.310.1 Function Documentation

struct channel_range* get_channel_range (char * *optarg*)

void process_cmd_line (struct opts * , int *argc,* char * *argv[]*)

void run_acquisition (AIOContinuousBuf * *buf,* struct opts * *options*)

AIOUSB_BOOL fnd (AIOUSBDevice * *dev*)

int main (int *argc,* char * *argv[]*)

Start with the NewAIOContinousBufFromJSON("{'aiocontin

34.310.2 Variable Documentation

FILE* fp

34.311 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

34.311.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.311.2 Variable Documentation

int CURRENT_DEBUG_LEVEL

34.312 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)

34.312.1 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice * *dev*)

int main (int *argc*, char ** *argv*)

34.313 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[])

34.313.1 Function Documentation

int main (int *argc*, char * *argv[]*)

34.314 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)

34.314.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.315 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)

34.315.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.316 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)

34.316.1 Function Documentation

int main (int *argc*, char ** *argv*)

34.317 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)

34.317.1 Function Documentation

int main (int *argc*, char ** *argv*)

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

< GetDevices

34.318 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)

34.318.1 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

34.318.2 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

34.318.3 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice * *dev*)

int main (int *argc*, char ** *argv*)

< GetDevices

34.319 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).

34.319.1 Function Documentation

int main (int *argc*, char ** *argv*)

LIBUSB Overview LIBUSB (http://www.libusb.org/) must be installed on the Linux box (the AIOUSB code was developed using libusb version 1.0.3).

After installing libusb, it may also be necessary to set an environment variable so that the libusb and aiousb header files can be located:

```
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aiousb/
```

Once libusb is installed properly, it should be possible to compile the sample program using the simple command:

```
make
```

Alternatively, one can "manually" compile the sample program using the command:

```
g++ sample.cpp -laiousb -lusb-1.0 -o sample
```

or, to enable debug features

```
g++ -ggdb sample.cpp -laiousbdbg -lusb-1.0 -o sample
```

< GetDevices

34.320 samples/USB-DIO-48/read_and_write.c File Reference

AIOUSB sample program that writes and reads from a USB-DIO-48.

```
#include "aiocommon.h"
#include <aiousb.h>
```

Functions

- AIOUSB_BOOL find_dio (AIOUSBDevice *dev)
- void CHECK_RESULT (AIORET_TYPE retval, char *errmsg)
- int main (int argc, char **argv)

34.320.1 Detailed Description

AIOUSB sample program that writes and reads from a USB-DIO-48.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

All the API functions that DO NOT begin "AIOUSB_" are standard API functions, largely documented in http://accesio.com/MANUALS/USB%20Software%20Reference.pdf. The functions that DO begin with "AIOUSB_" are "extended" API functions added to the Linux implementation. Source code lines in this sample program that are prefixed with the comment "/ ∗ API ∗ /" highlight calls to the AIOUSB API.

34.320.2 Function Documentation

AIOUSB_BOOL find_dio (AIOUSBDevice ∗ *dev*)

void CHECK_RESULT (AIORET_TYPE *retval,* char ∗ *errmsg*)

int main (int *argc,* char ∗∗ *argv*)

34.321 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)

34.321.1 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)))

34.321.2 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

34.322 samples/USB-DIO-96/mytest.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20
- #define PORT_C 1<<2
- #define PORT_B 1<<1
- #define PORT_A 1
- #define MAKE_MASK(GROUP, PORT) ((PORT << (GROUP * 3)))

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice *dev)
- char * show_byte (unsigned char)
- int main (int argc, char **argv)

34.322.1 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)))

34.322.2 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

34.323 samples/USB-DIO-96/tmp.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20
- #define PORT_C 1<<2
- #define PORT_B 1<<1
- #define PORT_A 1
- #define MAKE_MASK(GROUP, PORT) ((PORT << (GROUP ∗ 3)))

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- char ∗ show_byte (unsigned char)
- int main (int argc, char ∗∗argv)

34.323.1 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)))

34.323.2 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

34.324 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)

34.324.1 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

34.324.2 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

34.324.3 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice * *dev*)

AIOUSB_BOOL fnd (AIOUSBDevice * *dev*)

int main (int *argc*, char ** *argv*)

< GetDevices

34.325 samples/USB-IDIO-16_8/idio_sample.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "aiousb.h"
```

Macros

- #define RATE_LIMIT(product)

Functions

- int main (int argc, char *argv[])

34.325.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.325.2 Macro Definition Documentation

#define RATE_LIMIT(*product*)

Value:

```
do {                                                                    \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )           \
            sleep(1);                                                   \
    } while( 0 );                                                       \
```

34.325.3 Function Documentation

int main (int *argc*, char * *argv[]*)

34.326 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[])

34.326.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.326.2 Macro Definition Documentation

#define RATE_LIMIT(*product*)

Value:

```
do {                                                                    \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )           \
            sleep(1);                                                   \
    } while( 0 );                                                       \
```

34.326.3 Function Documentation

int main (int *argc*, char * *argv[]*)

34.327 samples/USB-IDIO-16_8/perftest.c File Reference

```
#include "aiousb.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
```

Functions

- int main (int argc, char *argv[])

  $Date

34.327.1 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

34.328 samples/USB-IIRO-16_8/iiro_sample.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "aiousb.h"
#include "aiocommon.h"
```

Macros

- #define RATE_LIMIT(product)

Functions

- AIOUSB_BOOL find_idio (AIOUSBDevice *dev)
- int main (int argc, char *argv[])

34.328.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

34.328.2 Macro Definition Documentation

#define RATE_LIMIT(*product*)

Value:

```
do {                                                                    \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )           \
            sleep(1);                                                   \
    } while( 0 );                                                       \
```

34.328.3 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

34.329 /media/jdamon/Development/Documents/Projects/aiousb_patrick_mcbride_issue/AIOUSB/README.md File Reference

34.330 lib/ndkbuild/libusb/README.md File Reference

34.331 lib/wrappers/README.md File Reference

34.332 samples/USB-AI16-16/android/read_channels_test/README.md File Reference

34.333 samples/USB-AI16-16/android/README.md File Reference

34.334 samples/USB-AI16-16/java/extcal/README.md File Reference

34.335 samples/USB-AI16-16/java/read_channels_test/native-utils/README.md File Reference

34.336 samples/USB-AI16-16/java/read_channels_test/README.md File Reference

34.337 samples/USB-AI16-16/java/README.md File Reference

Index

~AI16_DataSet
AIOUSB::AI16_DataSet, 149
~AI16_InputRange
AIOUSB::AI16_InputRange, 152
~AO16_AnalogOutputSubsystem
AIOUSB::AO16_AnalogOutputSubsystem, 193
~AO16_OutputRange
AIOUSB::AO16_OutputRange, 196
~AnalogIORange
AIOUSB::AnalogIORange, 189
~AnalogInputSubsystem
AIOUSB::AnalogInputSubsystem, 172
~AnalogOutputSubsystem
AIOUSB::AnalogOutputSubsystem, 191
~CounterSubsystem
AIOUSB::CounterSubsystem, 205
~DA12_AnalogOutputSubsystem
AIOUSB::DA12_AnalogOutputSubsystem, 208
~DA12_OutputRange
AIOUSB::DA12_OutputRange, 213
~DIOStreamSubsystem
AIOUSB::DIOStreamSubsystem, 226
~DeviceSubsystem
AIOUSB::DeviceSubsystem, 218
~DigitalIOSubsystem
AIOUSB::DigitalIOSubsystem, 220
~TestCaseSetup
TestCaseSetup, 273
~USBDevice
USBDevice, 313
~USBDeviceBase
AIOUSB::USBDeviceBase, 315
~USBDeviceHandle
USBDeviceHandle, 319
~USBDeviceManager
AIOUSB::USBDeviceManager, 321
~USBRoster
USBRoster, 334
~USBTransfer
USBTransfer, 335
~USB_AI16_Family
AIOUSB::USB_AI16_Family, 279
~USB_AIO16_Family
AIOUSB::USB_AIO16_Family, 282
~USB_AO16_Family
AIOUSB::USB_AO16_Family, 284
~USB_DIO_Family
AIOUSB::USB_DIO_Family, 301
~WatchedEntry
WatchedEntry, 336

ACCES_VENDOR_ID
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
AD_CAL_MODE_BIP_GROUND
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CAL_MODE_GROUND
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CAL_MODE_HIGH_REF
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CAL_MODE_NORMAL
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CAL_MODE_REFERENCE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CONFIG_CAL_MODE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_GAIN_CODE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_MUX_START_END
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_OVERSAMPLE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_REGISTERS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_CONFIG_START_END
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_START_STOP_CHANNEL_EX
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_CONFIG_TRIG_COUNT
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_DIFFERENTIAL_MODE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_0_10V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_0_1V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_0_2V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_0_5V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_10V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_1V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_2V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_5V
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_GAIN_CODE_MASK
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_MAX_CHANNELS
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_MAX_CONFIG_REGISTERS

AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_MAX_TIMEOUT
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_MIN_CONFIG_REGISTERS
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_MIN_TIMEOUT
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_MUX_CONFIG_REGISTERS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AD_NO_SET_CAL
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 414
AD_NUM_GAIN_CODE_REGISTERS
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_NUM_GAIN_CODES
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_REGISTER_CAL_MODE
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_REGISTER_GAIN_CODE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_REGISTER_MUX_START_END
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_REGISTER_OVERSAMPLE
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_REGISTER_START_END
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_REGISTER_TRIG_COUNT
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AD_SCAN_BULKACQUIRE
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SCAN_CONTINUOUS
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SCAN_GETCHANNEL
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SCAN_GETCHANNELV
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SCAN_GETSCAN
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SCAN_GETSCANV
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413, 414
AD_SET_CAL_AUTO
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 414
AD_SET_CAL_MANUAL
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 414
AD_SET_CAL_NORMAL
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 414
AD_TRIGGER_CTR0_EXT
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_TRIGGER_EXTERNAL
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_TRIGGER_FALLING_EDGE
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_TRIGGER_SCAN
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_TRIGGER_TIMER
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
AD_TRIGGER_VALID_MASK
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
ADCCONFIG_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
ADCCONIGBLOCK_CONFIG
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413
ADCalMode_begin
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
ADCalMode_end
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
ADGainCode_begin
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
ADGainCode_end
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
ADRegister_begin
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
ADRegister_end
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AI_16_MAX_COUNTS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIO_CONT_BUF_TYPE_COUNTS
AIOContinuousBuffer.h, 437
ndkbuild/jni/AIOContinuousBuffer.h, 445
AIO_CONT_BUF_TYPE_VOLTS
AIOContinuousBuffer.h, 437
ndkbuild/jni/AIOContinuousBuffer.h, 445
AIO_CONTINUE_RUNNING
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIO_COUNTS_BUF
AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIO_DEFAULT_BUF
AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIO_ERROR_BUF
AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIO_PER_CHANNEL
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIO_PER_OVERSAMPLE
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIO_PER_SCANS
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIO_TERMINATE_CALLBACK
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIO_VOLTS_BUF

AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIOBuf.h
AIO_COUNTS_BUF, 377
AIO_DEFAULT_BUF, 377
AIO_ERROR_BUF, 377
AIO_VOLTS_BUF, 377
AIOCONTINUOUS_BUF_ALLORNONE
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOCONTINUOUS_BUF_NORMAL
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOCONTINUOUS_BUF_OVERRIDE
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOCONTINUOUSBUF_CONFIG
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413
AIOChannelRange.c
BEGIN, 391
END, 391
END_CHANNEL, 391
GAIN, 391
START_CHANNEL, 391
AIOCommandCode_begin
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIOCommandCode_end
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIOCommandLine.h
ADCCONFIG_OPT, 404
CHANNEL_OPT, 404
COUNT_OPT, 404
DEBUG_OPT, 404
FILE_OPT, 404
INDEX_NUM, 404
SAMPLE_OPT, 404
SETCAL_OPT, 404
TIMEOUT_OPT, 404
AIOConfiguration.h
AD_NO_SET_CAL, 412
AD_SCAN_BULKACQUIRE, 412
AD_SCAN_CONTINUOUS, 412
AD_SCAN_GETCHANNEL, 412
AD_SCAN_GETCHANNELV, 412
AD_SCAN_GETSCAN, 412
AD_SCAN_GETSCANV, 412
AD_SET_CAL_AUTO, 412
AD_SET_CAL_MANUAL, 412
AD_SET_CAL_NORMAL, 412
ADCCONIGBLOCK_CONFIG, 412
AIOCONTINUOUSBUF_CONFIG, 412
NO_CONFIG, 412
AIOContinuousBufMode_begin
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOContinuousBufMode_end
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOContinuousBuffer.h
AIO_CONT_BUF_TYPE_COUNTS, 437
AIO_CONT_BUF_TYPE_VOLTS, 437
AIODEBUG_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIODEFAULT_LOG_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIODEVEL_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOERROR_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOEither.h
aioeither_value_double, 487
aioeither_value_double_t, 487
aioeither_value_int, 487
aioeither_value_int32_t, 487
aioeither_value_longdouble_t, 487, 488
aioeither_value_obj, 487, 488
aioeither_value_string, 487, 488
aioeither_value_uint16_t, 487
aioeither_value_uint32_t, 487
aioeither_value_uint8_t, 487
aioeither_value_unsigned, 487
aioeither_vlaue_int16_t, 487
AIOFATAL_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOINFO_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOTypes.h
ACCES_VENDOR_ID, 521, 523
AD_CAL_MODE_BIP_GROUND, 533
AD_CAL_MODE_GROUND, 533
AD_CAL_MODE_HIGH_REF, 533
AD_CAL_MODE_NORMAL, 533
AD_CAL_MODE_REFERENCE, 533
AD_CONFIG_CAL_MODE, 530
AD_CONFIG_GAIN_CODE, 530
AD_CONFIG_MUX_START_END, 530
AD_CONFIG_OVERSAMPLE, 530
AD_CONFIG_REGISTERS, 533
AD_CONFIG_START_END, 530
AD_CONFIG_START_STOP_CHANNEL_EX, 530
AD_CONFIG_TRIG_COUNT, 530
AD_DIFFERENTIAL_MODE, 530
AD_GAIN_CODE_0_10V, 530
AD_GAIN_CODE_0_1V, 530
AD_GAIN_CODE_0_2V, 530
AD_GAIN_CODE_0_5V, 530
AD_GAIN_CODE_10V, 530
AD_GAIN_CODE_1V, 530
AD_GAIN_CODE_2V, 530
AD_GAIN_CODE_5V, 530
AD_GAIN_CODE_MASK, 529
AD_MAX_CHANNELS, 529
AD_MAX_CONFIG_REGISTERS, 530
AD_MAX_TIMEOUT, 530
AD_MIN_CONFIG_REGISTERS, 530
AD_MIN_TIMEOUT, 530
AD_MUX_CONFIG_REGISTERS, 533
AD_NUM_GAIN_CODE_REGISTERS, 530
AD_NUM_GAIN_CODES, 530
AD_REGISTER_CAL_MODE, 529
AD_REGISTER_GAIN_CODE, 530
AD_REGISTER_MUX_START_END, 529
AD_REGISTER_OVERSAMPLE, 529
AD_REGISTER_START_END, 529
AD_REGISTER_TRIG_COUNT, 529
AD_TRIGGER_CTR0_EXT, 530
AD_TRIGGER_EXTERNAL, 530
AD_TRIGGER_FALLING_EDGE, 530
AD_TRIGGER_SCAN, 530
AD_TRIGGER_TIMER, 530
AD_TRIGGER_VALID_MASK, 530
ADCalMode_begin, 533
ADCalMode_end, 533
ADGainCode_begin, 530
ADGainCode_end, 530

ADRegister_begin, 529
ADRegister_end, 529
AI_16_MAX_COUNTS, 533
AIO_CONTINUE_RUNNING, 533
AIO_PER_CHANNEL, 520
AIO_PER_OVERSAMPLE, 520
AIO_PER_SCANS, 520
AIO_TERMINATE_CALLBACK, 533
AIOCONTINUOUS_BUF_ALLORNONE, 521
AIOCONTINUOUS_BUF_NORMAL, 521
AIOCONTINUOUS_BUF_OVERRIDE, 521
AIOCommandCode_begin, 533
AIOCommandCode_end, 533
AIOContinuousBufMode_begin, 521
AIOContinuousBufMode_end, 521
AIOUSB_ERROR_AIOCOMMANDLINE_HELP, 528, 529
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RANGE, 528, 529
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_INDEX_NUM, 528, 529
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS, 528, 529
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL, 528, 529
AIOUSB_ERROR_BAD_TOKEN_TYPE, 527, 528
AIOUSB_ERROR_DEVICE_NOT_CONNECTED, 527, 528
AIOUSB_ERROR_DEVICE_NOT_FOUND, 527, 528
AIOUSB_ERROR_DIVIDE_BY_ZERO, 527, 528
AIOUSB_ERROR_DUP_NAME, 527, 528
AIOUSB_ERROR_FILE_NOT_FOUND, 527, 528
AIOUSB_ERROR_HANDLE_EOF, 527, 528
AIOUSB_ERROR_INVALID_ADCCONFIG, 527, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING, 527, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_CHANNEL_SETTING, 528, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_DEVICE, 528, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING, 528, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING, 528, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_REGISTER_SETTING, 528, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_SETTING, 527, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE, 527, 529
AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING, 527, 529
AIOUSB_ERROR_INVALID_AIOARGUMENT, 527, 529
AIOUSB_ERROR_INVALID_AIOBUFTYPE, 528, 529
AIOUSB_ERROR_INVALID_AIOCHANNELMASK, 527, 528
AIOUSB_ERROR_INVALID_AIOCMD, 527, 528
AIOUSB_ERROR_INVALID_AIOCONFIGURATION, 527, 529
AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER, 528, 529
AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER_NUM_CHANNELS, 528, 529
AIOUSB_ERROR_INVALID_AIODEVICE_QUERY, 527, 529
AIOUSB_ERROR_INVALID_AIOEITHER, 527, 529
AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION, 527, 528
AIOUSB_ERROR_INVALID_AIOFIFO, 527, 529
AIOUSB_ERROR_INVALID_CALLBACK, 527, 528
AIOUSB_ERROR_INVALID_CALMODE, 527, 529
AIOUSB_ERROR_INVALID_CHANNEL_NUMBER, 527, 529
AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING, 527, 528
AIOUSB_ERROR_INVALID_CONFIG, 527, 529
AIOUSB_ERROR_INVALID_COUNTS, 527, 528
AIOUSB_ERROR_INVALID_COUNTS_CONVERTER, 527, 528
AIOUSB_ERROR_INVALID_DATA, 527, 528
AIOUSB_ERROR_INVALID_DEVICE, 527, 528
AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING, 527, 528
AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER, 527, 528
AIOUSB_ERROR_INVALID_DEVICE_MUX_CHANNEL_SETTING, 527, 528
AIOUSB_ERROR_INVALID_DEVICE_SETTING, 527, 528
AIOUSB_ERROR_INVALID_DEVICE_STREAM_SETTING, 527, 528
AIOUSB_ERROR_INVALID_DIOBUF, 527, 529
AIOUSB_ERROR_INVALID_GAINCODE, 527, 529
AIOUSB_ERROR_INVALID_INDEX, 527, 528
AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE, 528, 529
AIOUSB_ERROR_INVALID_MEMORY, 527, 528
AIOUSB_ERROR_INVALID_MUTEX, 527, 528
AIOUSB_ERROR_INVALID_PARAMETER, 527, 528
AIOUSB_ERROR_INVALID_THREAD, 527, 528
AIOUSB_ERROR_INVALID_TIMEOUT, 527, 528
AIOUSB_ERROR_INVALID_USBDEVICE, 527, 528
AIOUSB_ERROR_INVALID_VOLTAGES, 527, 528
AIOUSB_ERROR_LIBUSB, 528, 529
AIOUSB_ERROR_NOT_ENOUGH_MEMORY, 527, 528
AIOUSB_ERROR_NOT_INIT, 527, 528
AIOUSB_ERROR_NOT_SUPPORTED, 527, 528
AIOUSB_ERROR_OPEN_FAILED, 527, 528
AIOUSB_ERROR_TIMEOUT, 527, 528
AIOUSB_ERROR_USB_INIT, 527, 528
AIOUSB_ERROR_USBDEVICE_NOT_FOUND, 527, 528
AIOUSB_FALSE, 521
AIOUSB_FIFO_COPY_ERROR, 528, 529
AIOUSB_MAX_NAME_SIZE, 533
AIOUSB_SUCCESS, 527, 528
AIOUSB_TRUE, 521
AUR_ADC_GET_CONFIG, 531, 532
AUR_ADC_IMMEDIATE, 531, 532
AUR_ADC_SET_CONFIG, 531, 532
AUR_CTR_COS_BULK_ABORT, 531, 532
AUR_CTR_COS_BULK_GATE2, 531, 532
AUR_CTR_LOAD, 531, 532
AUR_CTR_MODE, 531, 532
AUR_CTR_MODELOAD, 531, 532
AUR_CTR_PUR_EVCT, 531, 532
AUR_CTR_PUR_FIRST, 531, 532
AUR_CTR_PUR_MFRQ, 531, 532
AUR_CTR_PUR_MPUL, 531, 532
AUR_CTR_PUR_OFRQ, 531, 532
AUR_CTR_READ, 531, 532
AUR_CTR_READALL, 531, 532
AUR_CTR_READLATCHED, 531, 532
AUR_CTR_SELGATE, 531, 532
AUR_DAC_CONTROL, 531, 532
AUR_DAC_DATAPTR, 531, 532
AUR_DAC_DIVISOR, 531, 532
AUR_DAC_IMMEDIATE, 531, 532
AUR_DAC_RANGE, 531, 532
AUR_DIO_CONFIG, 531, 532

AUR_DIO_CONFIG_QUERY, 531, 532
AUR_DIO_READ, 531, 532
AUR_DIO_SETCLOCKS, 531, 532
AUR_DIO_SPI_READ, 531, 532
AUR_DIO_SPI_WRITE, 531, 532
AUR_DIO_STREAM_OPEN_INPUT, 531, 532
AUR_DIO_STREAM_OPEN_OUTPUT, 531, 532
AUR_DIO_WDG16_DEPREC, 531, 532
AUR_DIO_WRITE, 531
AUR_EEPROM_READ, 531, 532
AUR_EEPROM_WRITE, 531, 532
AUR_FLASH_READWRITE, 531, 532
AUR_GEN_ABORT_AND_CLEAR, 531, 532
AUR_GEN_CLEAR_FIFO, 531, 532
AUR_GEN_CLEAR_FIFO_NEXT, 531, 532
AUR_GEN_CLEAR_FIFO_WAIT, 531, 532
AUR_GEN_STREAM_STATUS, 531, 532
AUR_LOAD_BULK_CALIBRATION_BLOCK, 531, 532
AUR_OFFLINE_READWRITE, 531, 532
AUR_PROBE_CALFEATURE, 531, 532
AUR_READBACK_GLOBAL_STATE, 531, 532
AUR_SAVE_GLOBAL_STATE, 531, 532
AUR_SELF_TEST_1, 531, 532
AUR_START_ACQUIRING_BLOCK, 531, 532
AUR_WDG, 531, 532
AUR_WDG_STATUS, 531, 532
BITS_PER_BYTE, 533
CAL_TABLE_WORDS, 533
CLEAR_FIFO_METHOD_AUTO, 526
CLEAR_FIFO_METHOD_IMMEDIATE, 526
CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT, 526
CLEAR_FIFO_METHOD_NOW, 526
CLEAR_FIFO_METHOD_WAIT, 526
COUNTER_NUM_MODES, 533
COUNTERS_PER_BLOCK, 533
CYPRESS_DESC_PARAMS, 533
CYPRESS_GET_DESC, 531, 532
CYPRESS_MAX_DESC_SIZE, 533
DAC_RANGE_0_10V, 526
DAC_RANGE_0_5V, 526
DAC_RANGE_10V, 526
DAC_RANGE_5V, 526
DAC_RESET, 533
DACRange_begin, 526
DACRange_end, 526
diFirst, 525
diNone, 525
diOnly, 525
EEPROM_CUSTOM_BASE_ADDRESS, 533
EEPROM_CUSTOM_MAX_ADDRESS, 533
EEPROM_CUSTOM_MIN_ADDRESS, 533
EEPROM_SERIAL_NUMBER_ADDRESS, 533
FIFO_Method_begin, 526
FIFO_Method_end, 526
GENERIC_DOSOMETHING_PLACEHOLDER, 533
INVALID_OBJECT, 520
JOINED, 520
MAX_IMM_ADCS, 533
MAX_USB_DEVICES, 521
NOT_STARTED, 520
PICO_DIO16RO8, 522, 524
ProductIDS_begin, 521, 523
ProductIDS_end, 523, 525
RUNNING, 520
RUNNING_OR_WITH_DATA, 520
ResultCode_begin, 527, 528
ResultCode_end, 528, 529
TERMINATED, 520
TERMINATED_OVERRUN, 520
TERMINATING, 520
THREAD_STATUS_begin, 520
THREAD_STATUS_end, 520
USB_AI12_128, 522, 524
USB_AI12_128A, 522, 524
USB_AI12_128E, 522, 524
USB_AI12_16, 522, 524
USB_AI12_16A, 522, 524
USB_AI12_16E, 522, 524
USB_AI12_32, 522, 524
USB_AI12_32A, 522, 524
USB_AI12_32E, 522, 524
USB_AI12_64, 522, 524
USB_AI12_64A, 522, 524
USB_AI12_64E, 522, 524
USB_AI12_64M, 522, 524
USB_AI12_64MA, 522, 524
USB_AI12_64ME, 522, 524
USB_AI12_96, 522, 524
USB_AI12_96A, 522, 524
USB_AI12_96E, 522, 524
USB_AI16_128A, 522, 524
USB_AI16_128E, 522, 524
USB_AI16_16A, 522, 524
USB_AI16_16E, 522, 524
USB_AI16_32A, 522, 524
USB_AI16_32E, 522, 524
USB_AI16_64A, 522, 524
USB_AI16_64E, 522, 524
USB_AI16_64MA, 522, 524
USB_AI16_64ME, 522, 524
USB_AI16_96A, 522, 524
USB_AI16_96E, 522, 524
USB_AIO12_128, 523, 525
USB_AIO12_128A, 523, 525
USB_AIO12_128E, 523, 525
USB_AIO12_16, 523, 525
USB_AIO12_16A, 523, 525
USB_AIO12_16E, 523, 525
USB_AIO12_32, 523, 525
USB_AIO12_32A, 523, 525
USB_AIO12_32E, 523, 525
USB_AIO12_64, 523, 525
USB_AIO12_64A, 523, 525
USB_AIO12_64E, 523, 525
USB_AIO12_64M, 523, 525
USB_AIO12_64MA, 523, 525
USB_AIO12_64ME, 523, 525
USB_AIO12_96, 523, 525
USB_AIO12_96A, 523, 525
USB_AIO12_96E, 523, 525
USB_AIO16_128A, 523, 525
USB_AIO16_128E, 523, 525
USB_AIO16_16A, 523, 525
USB_AIO16_16E, 523, 525
USB_AIO16_32A, 523, 525
USB_AIO16_32E, 523, 525
USB_AIO16_64A, 523, 525
USB_AIO16_64E, 523, 525
USB_AIO16_64MA, 523, 525
USB_AIO16_64ME, 523, 525
USB_AIO16_96A, 523, 525
USB_AIO16_96E, 523, 525
USB_AO12_12, 523, 525
USB_AO12_12A, 523, 525
USB_AO12_16, 523, 525
USB_AO12_16A, 522, 525
USB_AO12_4, 523, 525
USB_AO12_4A, 523, 525
USB_AO12_8, 523, 525
USB_AO12_8A, 523, 525
USB_AO16_12, 522, 525
USB_AO16_12A, 522, 524

USB_AO16_16, 522, 524
USB_AO16_16A, 522, 524
USB_AO16_4, 522, 525
USB_AO16_4A, 522, 525
USB_AO16_8, 522, 525
USB_AO16_8A, 522, 525
USB_AO_ARB1, 522, 524
USB_BULK_READ_ENDPOINT, 533
USB_BULK_WRITE_ENDPOINT, 533
USB_CTR_15, 522, 524
USB_DA12_8A, 521, 523
USB_DA12_8A_REV_A, 521, 523
USB_DA12_8E, 521, 523
USB_DI16A, 521, 523
USB_DI16A_REV_A1, 521, 523
USB_DI16A_REV_A2, 521, 523
USB_DIO16RO8, 522, 524
USB_DIO24_CTR6, 521, 523
USB_DIO24DO12, 522, 524
USB_DIO48DO24, 522, 524
USB_DIO_16A, 521, 523
USB_DIO_16H, 521, 523
USB_DIO_32, 521, 523
USB_DIO_32I, 521, 523
USB_DIO_48, 521, 523
USB_DIO_96, 521, 523
USB_DO16A, 521, 523
USB_DO16A_REV_A1, 521, 523
USB_DO24, 522, 524
USB_IDIO_16, 522, 524
USB_IDIO_4, 522, 524
USB_IDIO_8, 522, 524
USB_IDO_16, 522, 524
USB_II_16, 521, 524
USB_II_16_OLD, 522, 524
USB_II_8, 521, 524
USB_II_8_OLD, 522, 524
USB_IIRO4_2SM, 522, 524
USB_IIRO4_COM, 522, 524
USB_IIRO_16, 521, 524
USB_IIRO_4, 521, 524
USB_IIRO_8, 521, 524
USB_READ_FROM_DEVICE, 533
USB_RO_16, 521, 524
USB_WRITE_TO_DEVICE, 533
USBP_II8IDO4A, 522, 524
VENDOR_REQUEST_begin, 531
VENDOR_REQUEST_end, 531, 532
WITH_DATA, 520
AIOUSB_ERROR_AIOCOMMANDLINE_HELP
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RANGE
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_INDEX_NUM
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_BAD_TOKEN_TYPE
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_DEVICE_NOT_CONNECTED
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_DEVICE_NOT_FOUND
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_DIVIDE_BY_ZERO
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_DUP_NAME
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_FILE_NOT_FOUND
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_HANDLE_EOF
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_ADCCONFIG
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_CHANNEL_SETTING
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_DEVICE
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_REGISTER_SETTING
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_SETTING
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_AIOARGUMENT
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_AIOBUFTYPE
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_AIOCHANNELMASK
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_AIOCMD
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_AIOCONFIGURATION
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551

AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER_NUM_CHANNELS
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_AIODEVICE_QUERY
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_AIOEITHER
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_AIOFIFO
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_CALLBACK
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_CALMODE
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_CHANNEL_NUMBER
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_CONFIG
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_COUNTS
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_COUNTS_CONVERTER
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DATA
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_MUX_CHANNEL_SETTING
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_SETTING
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_STREAM_SETTING
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_DIOBUF
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_GAINCODE
AIOTypes.h, 527, 529
ndkbuild/jni/AIOTypes.h, 549, 551
AIOUSB_ERROR_INVALID_INDEX
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_INVALID_MEMORY
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_MUTEX
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_PARAMETER
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_THREAD
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_TIMEOUT
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_USBDEVICE
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_INVALID_VOLTAGES
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_LIBUSB
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_ERROR_NOT_ENOUGH_MEMORY
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_NOT_INIT
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_NOT_SUPPORTED
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_OPEN_FAILED
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_TIMEOUT
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_USB_INIT
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_ERROR_USBDEVICE_NOT_FOUND
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_FALSE
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOUSB_FIFO_COPY_ERROR
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
AIOUSB_Log.h
AIODEBUG_LEVEL, 648
AIODEFAULT_LOG_LEVEL, 648
AIODEVEL_LEVEL, 648
AIOERROR_LEVEL, 648
AIOFATAL_LEVEL, 648
AIOINFO_LEVEL, 648
AIOWARN_LEVEL, 648
AIOUSB_MAX_NAME_SIZE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIOUSB_Properties.h
BASIC, 655
JSON, 655
TERSE, 655
YAML, 655
AIOUSB_SUCCESS
AIOTypes.h, 527, 528

ndkbuild/jni/AIOTypes.h, 549, 550
AIOUSB_TRUE
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOUSB_WDG.h
AIOUSB_WDG_READ_INDEX, 663
AIOUSB_WDG_READ_VALUE, 663
WDGVals_begin, 663
WDGVals_end, 663
AIOUSB_WDG_READ_INDEX
AIOUSB_WDG.h, 663
ndkbuild/jni/AIOUSB_WDG.h, 664
AIOUSB_WDG_READ_VALUE
AIOUSB_WDG.h, 663
ndkbuild/jni/AIOUSB_WDG.h, 664
AIOWARN_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AUR_ADC_GET_CONFIG
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_ADC_IMMEDIATE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_ADC_SET_CONFIG
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_COS_BULK_ABORT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_COS_BULK_GATE2
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_LOAD
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_MODE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_MODELOAD
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_PUR_EVCT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_PUR_FIRST
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_PUR_MFRQ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_PUR_MPUL
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_PUR_OFRQ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_READ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_READALL
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_READLATCHED
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_CTR_SELGATE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DAC_CONTROL
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DAC_DATAPTR
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DAC_DIVISOR
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DAC_IMMEDIATE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DAC_RANGE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_CONFIG
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_CONFIG_QUERY
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_READ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_SETCLOCKS
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_SPI_READ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_SPI_WRITE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_STREAM_OPEN_INPUT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_STREAM_OPEN_OUTPUT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_WDG16_DEPREC
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_DIO_WRITE
AIOTypes.h, 531
ndkbuild/jni/AIOTypes.h, 553
AUR_EEPROM_READ
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_EEPROM_WRITE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_FLASH_READWRITE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_GEN_ABORT_AND_CLEAR
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_GEN_CLEAR_FIFO
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_GEN_CLEAR_FIFO_NEXT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_GEN_CLEAR_FIFO_WAIT
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_GEN_STREAM_STATUS
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_LOAD_BULK_CALIBRATION_BLOCK
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_OFFLINE_READWRITE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554

AUR_PROBE_CALFEATURE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_READBACK_GLOBAL_STATE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_SAVE_GLOBAL_STATE
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_SELF_TEST_1
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_START_ACQUIRING_BLOCK
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_WDG
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
AUR_WDG_STATUS
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
ACCES_DEPRECATED
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
ADBuf
AIOUSBDevice, 168
ADBuf_size
AIOUSBDevice, 168
ADC_ADMode
AIOUSB_ADC.c, 562
AIOUSB_ADC.h, 590
libaiousb.h, 735
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 602
ADC_Acquire_Reference_Counts
AIOUSB_ADC.c, 559
ndkbuild/jni/AIOUSB_ADC.c, 572
ADC_BulkAcquire
AIOUSB_ADC.c, 564
AIOUSB_ADC.h, 591
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 578
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_BulkPoll
AIOUSB_ADC.c, 565
AIOUSB_ADC.h, 591
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_CanCalibrate
AIOUSB_ADC.c, 564
AIOUSB_ADC.h, 591
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 578
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_ClearADBuf
AIOUSB_ADC.c, 565
ndkbuild/jni/AIOUSB_ADC.c, 579
ADC_ClearFastITConfig
AIOUSB_ADC.c, 565
ndkbuild/jni/AIOUSB_ADC.c, 579
ADC_ClockRateForADCProduct
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 590
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_CopyConfig
AIOUSB_ADC.c, 562
ndkbuild/jni/AIOUSB_ADC.c, 575
ADC_CreateADBuf
AIOUSB_ADC.c, 565
ndkbuild/jni/AIOUSB_ADC.c, 579
ADC_CreateFastITConfig
AIOUSB_ADC.c, 565
AIOUSB_ADC.h, 591
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_Debug_Register_Settings
AIOUSB_ADC.c, 565
ndkbuild/jni/AIOUSB_ADC.c, 580
ADC_GetADConfigBlock_Registers
AIOUSB_ADC.c, 565
AIOUSB_ADC.h, 598
libaiousb.h, 742
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 609
ADC_GetChannelV
AIOUSB_ADC.c, 560
AIOUSB_ADC.h, 586
libaiousb.h, 733
ndkbuild/jni/AIOUSB_ADC.c, 574
ndkbuild/jni/AIOUSB_ADC.h, 600
ADC_GetConfig
AIOUSB_ADC.c, 561
AIOUSB_ADC.h, 589
libaiousb.h, 734
ndkbuild/jni/AIOUSB_ADC.c, 575
ndkbuild/jni/AIOUSB_ADC.h, 602
ADC_GetConfigRegisters
AIOUSB_ADC.c, 559
ndkbuild/jni/AIOUSB_ADC.c, 572
ADC_GetConfigSize
AIOUSB_ADC.c, 559
ndkbuild/jni/AIOUSB_ADC.c, 572
ADC_GetFastITScanV
AIOUSB_ADC.c, 566
AIOUSB_ADC.h, 592
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 580
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_GetITScanV
AIOUSB_ADC.c, 566
AIOUSB_ADC.h, 592
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 580
ndkbuild/jni/AIOUSB_ADC.h, 605
ADC_GetMaxClockRate
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 590
libaiousb.h, 735
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_GetOversample
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 592
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 605
ADC_GetScan
AIOUSB_ADC.c, 561
AIOUSB_ADC.h, 588
libaiousb.h, 733
ndkbuild/jni/AIOUSB_ADC.c, 574
ndkbuild/jni/AIOUSB_ADC.h, 601
ADC_GetScanV
AIOUSB_ADC.c, 561
AIOUSB_ADC.h, 586
libaiousb.h, 732
ndkbuild/jni/AIOUSB_ADC.c, 574
ndkbuild/jni/AIOUSB_ADC.h, 600
ADC_InitFastITScanV
AIOUSB_ADC.c, 565

AIOUSB_ADC.h, 591
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_Initialize
AIOUSB_ADC.c, 564
AIOUSB_ADC.h, 591
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 578
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_QueryCal
AIOUSB_ADC.c, 564
AIOUSB_ADC.h, 590
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 578
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_Range1
AIOUSB_ADC.c, 562
AIOUSB_ADC.h, 589
libaiousb.h, 735
ndkbuild/jni/AIOUSB_ADC.c, 575
ndkbuild/jni/AIOUSB_ADC.h, 602
ADC_RangeAll
AIOUSB_ADC.c, 562
AIOUSB_ADC.h, 586
libaiousb.h, 733
ndkbuild/jni/AIOUSB_ADC.c, 575
ndkbuild/jni/AIOUSB_ADC.h, 600
ADC_ReadADConfigBlock
AIOUSB_ADC.c, 559
ndkbuild/jni/AIOUSB_ADC.c, 572
ADC_ResetDevice
AIOUSB_ADC.c, 559
AIOUSB_Core.h, 616
libaiousb.h, 712
ndkbuild/jni/AIOUSB_ADC.c, 572
ndkbuild/jni/AIOUSB_Core.h, 620
ADC_ResetFastITScanV
AIOUSB_ADC.c, 565
AIOUSB_ADC.h, 592
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_SetAllGainCodeAndDiffMode
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 594
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 607
ADC_SetCal
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 590
libaiousb.h, 736
ndkbuild/jni/AIOUSB_ADC.c, 578
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_SetConfig
AIOUSB_ADC.c, 562
AIOUSB_ADC.h, 589
libaiousb.h, 735
ndkbuild/jni/AIOUSB_ADC.c, 575
ndkbuild/jni/AIOUSB_ADC.h, 602
ADC_SetFastITScanVChannels
AIOUSB_ADC.c, 565
AIOUSB_ADC.h, 592
libaiousb.h, 737
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 604
ADC_SetOversample
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 592
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 605
ADC_SetScanLimits
AIOUSB_ADC.c, 563
AIOUSB_ADC.h, 590
libaiousb.h, 735
ndkbuild/jni/AIOUSB_ADC.c, 577
ndkbuild/jni/AIOUSB_ADC.h, 603
ADC_VerifyAndCorrectConfigBlock
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 372
ADCChannels
AIOUSBDevice, 166
DeviceProperties, 218
ADCChannelsPerGroup
AIOUSBDevice, 166
DeviceProperties, 218
mux_settings, 260
ADCConfigBlock, 145
ADCConfigBlock.h, 367
clock_rate, 146
debug, 146
device, 145
discardFirstSample, 146
mux_settings, 146
ndkbuild/jni/ADCConfigBlock.h, 372
registers, 146
size, 145
testing, 146
timeout, 146
ADCConfigBlock.c
ADC_VerifyAndCorrectConfigBlock, 361
ADCConfigBlockCopy, 360
ADCConfigBlockGetAIOUSBDevice, 360
ADCConfigBlockGetCalMode, 361
ADCConfigBlockGetClockRate, 363
ADCConfigBlockGetDebug, 361
ADCConfigBlockGetEndChannel, 362
ADCConfigBlockGetGainCode, 361
ADCConfigBlockGetJSONValueOrDefault, 362
ADCConfigBlockGetJSONValueOrInt, 362
ADCConfigBlockGetOversample, 362
ADCConfigBlockGetSize, 361
ADCConfigBlockGetStartChannel, 361
ADCConfigBlockGetTesting, 361
ADCConfigBlockGetTimeout, 362
ADCConfigBlockGetTriggerMode, 362
ADCConfigBlockInit, 361
ADCConfigBlockInitForCounterScan, 361
ADCConfigBlockInitializeDefault, 360
ADCConfigBlockInitializeFromAIOUSBDevice, 360
ADCConfigBlockSetAIOUSBDevice, 360
ADCConfigBlockSetAllGainCodeAndDiffMode, 361
ADCConfigBlockSetCalMode, 361
ADCConfigBlockSetChannelRange, 361
ADCConfigBlockSetClockRate, 363
ADCConfigBlockSetDebug, 361
ADCConfigBlockSetDevice, 360
ADCConfigBlockSetDifferentialMode, 362
ADCConfigBlockSetEndChannel, 361
ADCConfigBlockSetGainCode, 361
ADCConfigBlockSetOversample, 362
ADCConfigBlockSetRangeSingle, 361
ADCConfigBlockSetReference, 362
ADCConfigBlockSetRegister, 361
ADCConfigBlockSetScanAllChannels, 362
ADCConfigBlockSetScanRange, 361
ADCConfigBlockSetSize, 360
ADCConfigBlockSetStartChannel, 361
ADCConfigBlockSetTesting, 361

ADCConfigBlockSetTimeout, 362
ADCConfigBlockSetTriggerEdge, 362
ADCConfigBlockSetTriggerMode, 362
ADCConfigBlockSetTriggerReference, 362
ADCConfigBlockToJSON, 362
ADCConfigBlockToYAML, 362
DeleteADCConfigBlock, 360
get_cal_mode, 362
get_gain_code, 362
is_all_digits, 362
NewADCConfigBlockFromJSON, 363
ADCConfigBlock.h
ADC_VerifyAndCorrectConfigBlock, 369
ADCConfigBlock, 367
ADCConfigBlockCopy, 370
ADCConfigBlockGetAIOUSBDevice, 370
ADCConfigBlockGetCalMode, 369
ADCConfigBlockGetClockRate, 369
ADCConfigBlockGetDebug, 370
ADCConfigBlockGetEndChannel, 370
ADCConfigBlockGetGainCode, 369
ADCConfigBlockGetOversample, 370
ADCConfigBlockGetSize, 370
ADCConfigBlockGetStartChannel, 370
ADCConfigBlockGetTesting, 370
ADCConfigBlockGetTimeout, 370
ADCConfigBlockGetTriggerMode, 370
ADCConfigBlockInit, 368
ADCConfigBlockInitForCounterScan, 369
ADCConfigBlockInitializeDefault, 369
ADCConfigBlockInitializeFromAIOUSBDevice, 370
ADCConfigBlockSetAIOUSBDevice, 370
ADCConfigBlockSetAllGainCodeAndDiffMode, 369
ADCConfigBlockSetCalMode, 369
ADCConfigBlockSetChannelRange, 369
ADCConfigBlockSetClockRate, 369
ADCConfigBlockSetDebug, 370
ADCConfigBlockSetDevice, 370
ADCConfigBlockSetDifferentialMode, 370
ADCConfigBlockSetEndChannel, 369
ADCConfigBlockSetGainCode, 369
ADCConfigBlockSetOversample, 370
ADCConfigBlockSetRangeSingle, 370
ADCConfigBlockSetReference, 370
ADCConfigBlockSetRegister, 369
ADCConfigBlockSetScanRange, 369
ADCConfigBlockSetSize, 370
ADCConfigBlockSetStartChannel, 369
ADCConfigBlockSetTesting, 370
ADCConfigBlockSetTimeout, 370
ADCConfigBlockSetTriggerEdge, 370
ADCConfigBlockSetTriggerMode, 370
ADCConfigBlockToJSON, 370
ADCConfigBlockToYAML, 369
ADCMuxSettings, 367
ADConfigBlock, 367
AIOUSBDevice, 367
DeleteADCConfigBlock, 371
is_all_digits, 371
NewADCConfigBlockFromJSON, 371
ADCConfigBlockCopy
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockGetAIOUSBDevice
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockGetCalMode
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetClockRate
ADCConfigBlock.c, 363
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 366
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetDebug
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockGetEndChannel
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetGainCode
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockGetJSONValueOrDefault
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 366
ADCConfigBlockGetJSONValueOrInt
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 366
ADCConfigBlockGetOversample
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetSize
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockGetStartChannel
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetTesting
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockGetTimeout
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockGetTriggerMode
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373

ADCConfigBlockInit
ADCConfigBlock.c, 361
ADCConfigBlock.h, 368
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockInitForCounterScan
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockInitializeDefault
ADCConfigBlock.c, 360
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockInitializeFromAIOUSBDevice
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetAIOUSBDevice
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetAllGainCodeAndDiffMode
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockSetCalMode
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetChannelRange
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetClockRate
ADCConfigBlock.c, 363
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 366
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockSetDebug
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetDevice
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetDifferentialMode
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetEndChannel
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetGainCode
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockSetOversample
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetRangeSingle
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetReference
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetRegister
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 721
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 372
ADCConfigBlockSetScanAllChannels
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 366
ADCConfigBlockSetScanRange
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetSize
ADCConfigBlock.c, 360
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetStartChannel
ADCConfigBlock.c, 361
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetTesting
ADCConfigBlock.c, 361
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetTimeout
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetTriggerEdge
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370

libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockSetTriggerMode
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfigBlockSetTriggerReference
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 366
ADCConfigBlockToJSON
ADCConfigBlock.c, 362
ADCConfigBlock.h, 370
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 366
ndkbuild/jni/ADCConfigBlock.h, 374
ADCConfigBlockToYAML
ADCConfigBlock.c, 362
ADCConfigBlock.h, 369
libaiousb.h, 722
ndkbuild/jni/ADCConfigBlock.c, 365
ndkbuild/jni/ADCConfigBlock.h, 373
ADCConfiguration
AIOConfiguration.h, 411
ndkbuild/jni/AIOConfiguration.h, 413
ADCMUXChannels
AIOUSBDevice, 166
DeviceProperties, 218
mux_settings, 260
ADCMuxSettings
ADCConfigBlock.h, 367
ndkbuild/jni/ADCConfigBlock.h, 372
ADCScanType
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413
ADCSetCalFunction
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 414
ADCalMode
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
ADConfigBlock
ADCConfigBlock.h, 367
ndkbuild/jni/ADCConfigBlock.h, 372
ADConfigBlockToYAML
AIOUSB_ADC.c, 565
ndkbuild/jni/AIOUSB_ADC.c, 580
ADGainCode
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
ADRange, 146
minVolts, 146
range, 146
ADRegister
AIOTypes.h, 529
ndkbuild/jni/AIOTypes.h, 551
AI16_DataPoint, 146
AIOUSB::AI16_DataPoint, 147
AIOUSB::AI16_InputRange, 152
AI16_DataPointArray, 148
AIOUSB::AI16_DataPoint, 148
AIOUSB::AI16_DataPointArray, 148
AI16_DataSet, 149
AIOUSB::AI16_DataPoint, 148
AIOUSB::AI16_DataSet, 149
AI16_InputRange, 151
AIOUSB::AI16_InputRange, 152
AIO_ASSERT
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AIO_ASSERT_CONFIG
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
AIO_ASSERT_DIOBUF
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
AIO_ASSERT_EXIT
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AIO_ASSERT_RET
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AIO_ASSERT_USB
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
AIO_CHAR_ARRAY
AIOTuple.h, 510
ndkbuild/jni/AIOTuple.h, 511
AIO_CMDLINE_OPTIONS
libaiousb.h, 747
AIO_DEBUG_LEVEL
AIOUSB_Log.h, 648
AIO_DEBUG_STR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 649
AIO_DEVEL_STR
AIOUSB_Log.h, 647
ndkbuild/jni/AIOUSB_Log.h, 649
AIO_EITHER_TYPE
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
AIO_ERROR
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
AIO_ERROR_STR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIO_ERROR_VALUE
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
AIO_FATAL_STR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIO_FIFO_INTERFACE
AIOFifo, 163
AIOFifo.h, 495
new_aio_fifo, 261
AIO_INFO_STR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIO_MAKE_ERROR
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AIO_NUMBER
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIO_OPTIONS
aiocommon.c, 849
aiocommon.h, 850
AIO_PRODUCT_GROUP
AIOProductTypes.h, 507
AIO_RANGE
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.h, 509
AIO_RESET_STR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIO_SCAN_TYPE
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIO_WARN_STR
AIOUSB_Log.h, 648

ndkbuild/jni/AIOUSB_Log.h, 649
AIOArgument, 154
actual_size, 155
config, 155
debug, 155
size, 155
threaded, 155
AIOArgumentInitialize
AIOConfiguration.c, 410
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.c, 411
ndkbuild/jni/AIOConfiguration.h, 414
AIOArguments, 155
device_args, 155
number_arguments, 155
AIOBuf, 155
AIOBuf.h, 377
defined, 155
endpos, 155
ndkbuild/jni/AIOBuf.h, 379
size, 155
type, 155
AIOBuf.c
AIOBufGetIterator, 375
AIOBufGetRaw, 375
AIOBufGetSize, 375
AIOBufGetTotalSize, 375
AIOBufGetType, 375
AIOBufGetTypeSize, 375
AIOBufIteratorGetValue, 375
AIOBufIteratorIsValid, 375
AIOBufIteratorNext, 375
AIOBufRead, 375
AIOBufWrite, 375
DeleteAIOBuf, 375
NewAIOBuf, 375
AIOBuf.h
AIOBuf, 377
AIOBufGetIterator, 377
AIOBufGetRaw, 377
AIOBufGetSize, 377
AIOBufGetType, 377
AIOBufIterator, 377
AIOBufIteratorGetValue, 377
AIOBufIteratorIsValid, 378
AIOBufIteratorNext, 378
AIOBufRead, 377
AIOBufType, 377
AIOBufWrite, 377
DeleteAIOBuf, 377
NewAIOBuf, 377
AIOBufGetIterator
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufGetRaw
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufGetSize
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufGetTotalSize
AIOBuf.c, 375
ndkbuild/jni/AIOBuf.c, 376
AIOBufGetType
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufGetTypeSize
AIOBuf.c, 375
ndkbuild/jni/AIOBuf.c, 376
AIOBufIterator
AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIOBufIteratorGetValue
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 726
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufIteratorIsValid
AIOBuf.c, 375
AIOBuf.h, 378
libaiousb.h, 726
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufIteratorNext
AIOBuf.c, 375
AIOBuf.h, 378
libaiousb.h, 726
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 380
AIOBufRead
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufType
AIOBuf.h, 377
ndkbuild/jni/AIOBuf.h, 379
AIOBufWrite
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
AIOBufferType
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIOChannelMask, 156
active_signals, 156
number_signals, 156
pos, 156
signal_index, 156
signal_indices, 156
signals, 156
size, 156
strrep, 156
strrepsmall, 156
AIOChannelMask.c
AIOChannelMaskGetMask, 382
AIOChannelMaskGetMaskAtIndex, 381
AIOChannelMaskGetSize, 381
AIOChannelMaskIndices, 381
AIOChannelMaskNextIndex, 381
AIOChannelMaskNumberChannels, 381
AIOChannelMaskNumberSignals, 381
AIOChannelMaskSetMaskAtIndex, 381
AIOChannelMaskSetMaskFromInt, 381
AIOChannelMaskSetMaskFromStr, 381
AIOChannelMaskToString, 382
AIOChannelMaskToStringAtIndex, 382
DeleteAIOChannelMask, 380

NewAIOChannelMask, 380
NewAIOChannelMaskFromChr, 382
NewAIOChannelMaskFromStr, 381
AIOChannelMask.h
AIOChannelMaskGetMask, 386
AIOChannelMaskGetMaskAtIndex, 386
AIOChannelMaskGetSize, 387
AIOChannelMaskIndices, 387
AIOChannelMaskNextIndex, 387
AIOChannelMaskNumberChannels, 386
AIOChannelMaskNumberSignals, 387
AIOChannelMaskSetMaskAtIndex, 387
AIOChannelMaskSetMaskFromInt, 387
AIOChannelMaskSetMaskFromStr, 387
AIOChannelMaskToString, 386
AIOChannelMaskToStringAtIndex, 386
aio_channel_obj, 385
BIT_LENGTH, 385
DeleteAIOChannelMask, 385
NewAIOChannelMask, 385
NewAIOChannelMaskFromChr, 386
NewAIOChannelMaskFromStr, 386
AIOChannelMaskGetMask
AIOChannelMask.c, 382
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 389
AIOChannelMaskGetMaskAtIndex
AIOChannelMask.c, 381
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 389
AIOChannelMaskGetSize
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskIndices
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskNextIndex
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskNumberChannels
AIOChannelMask.c, 381
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 389
AIOChannelMaskNumberSignals
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskSetMaskAtIndex
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskSetMaskFromInt
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskSetMaskFromStr
AIOChannelMask.c, 381
AIOChannelMask.h, 387
libaiousb.h, 725
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 390
AIOChannelMaskToString
AIOChannelMask.c, 382
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 389
AIOChannelMaskToStringAtIndex
AIOChannelMask.c, 382
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 389
AIOChannelRange
AIOChannelRange.h, 393
libaiousb.h, 708
ndkbuild/jni/AIOChannelRange.h, 394
AIOChannelRange.c
AIOChannelRangeGetEnd, 391
AIOChannelRangeGetGain, 391
AIOChannelRangeGetStart, 391
AIOChannelRangeToStr, 391
aio_channel_range_error, 391
DeleteAIOChannelRange, 391
lookup_voltage_range, 391
NewAIOChannelRangeFromStr, 391
STATE, 391
AIOChannelRange.h
AIOChannelRange, 393
AIOChannelRangeGetEnd, 393
AIOChannelRangeGetGain, 393
AIOChannelRangeGetStart, 393
AIOChannelRangeToStr, 393
aio_channel_range_error, 393
DeleteAIOChannelRange, 393
NewAIOChannelRangeFromStr, 393
AIOChannelRangeGetEnd
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
AIOChannelRangeGetGain
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
AIOChannelRangeGetStart
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
AIOChannelRangeTmp, 157
AIOCommandLine.h, 404
end_channel, 157
gaincode, 157
ndkbuild/jni/AIOCommandLine.h, 407
start_channel, 157
AIOChannelRangeToStr
AIOChannelRange.c, 391
AIOChannelRange.h, 393

libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
AIOCmd, 157
AIOCmd.h, 396
channel, 157
ndkbuild/jni/AIOCmd.h, 396
num_channels, 157
num_samples, 157
num_scans, 157
stop_scan, 157
stop_scan_arg, 157
AIOCmd.c
DeleteAIOCmd, 395
NewAIOCmd, 395
NewAIOCmdFromJSON, 395
AIOCmd.h
AIOCmd, 396
DeleteAIOCmd, 396
NewAIOCmd, 396
NewAIOCmdFromJSON, 396
AIOCommandCode
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
AIOCommandLine.c
AIOCommandLineListDevices, 398
AIOCommandLineOptionsListDevices, 398
AIOCommandLineOptionsOverrideADCConfigBlock, 398
AIOCommandLineOverrideADCConfigBlock, 398
AIOGetChannelRange, 398
AIOPrintUsage, 397
AIOProcessCmdline, 397
AIOProcessCommandLine, 397
DeleteAIOCommandLineOptions, 398
NewAIOCommandLineOptionsFromDefaultOptions, 398
NewDefaultAIOCommandLineOptions, 398
opterr, 399
optind, 399
AIOCommandLine.h
AIOChannelRangeTmp, 404
AIOCommandLineListDevices, 406
AIOCommandLineOptions, 404
AIOCommandLineOptionsListDevices, 405
AIOCommandLineOptionsOverrideADCConfigBlock, 405
AIOCommandLineOverrideADCConfigBlock, 406
AIOGetChannelRange, 405
AIOPrintUsage, 405
AIOProcessCmdline, 404
AIOProcessCommandLine, 405
CNTS, 404
DUMP, 404
DeleteAIOCommandLineOptions, 405
DeviceEnum, 404
JCONF, 404
NewAIOCommandLineOptionsFromDefaultOptions, 404
NewDefaultAIOCommandLineOptions, 404
REPEAT, 404
AIOCommandLineListDevices
AIOCommandLine.c, 398
AIOCommandLine.h, 406
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 409
AIOCommandLineOptions, 157
adcconfig_json, 159
aiobuf_json, 159
AIOCommandLine.h, 404
block_size, 159
buffer_size, 159
calibration, 159
clock_rate, 158
counts, 159
debug_level, 158
default_aiobuf_json, 159
default_clock_rate, 158
default_end_channel, 159
default_num_channels, 158
default_num_oversamples, 158
default_num_scans, 158
default_start_channel, 159
end_channel, 159
gain_code, 158
index, 159
ndkbuild/jni/AIOCommandLine.h, 407
num_channels, 158
num_oversamples, 158
num_scans, 158
number_ranges, 158
outfile, 158
pass_through, 158
physical, 159
ranges, 159
rate_limit, 159
repeat_number, 159
reset, 158
slow_acquire, 159
start_channel, 158
verbose, 158
with_timing, 159
AIOCommandLineOptionsListDevices
AIOCommandLine.c, 398
AIOCommandLine.h, 405
libaiousb.h, 747
ndkbuild/jni/AIOCommandLine.c, 401
ndkbuild/jni/AIOCommandLine.h, 409
AIOCommandLineOptionsOverrideADCConfigBlock
AIOCommandLine.c, 398
AIOCommandLine.h, 405
libaiousb.h, 747
ndkbuild/jni/AIOCommandLine.c, 402
ndkbuild/jni/AIOCommandLine.h, 409
AIOCommandLineOverrideADCConfigBlock
AIOCommandLine.c, 398
AIOCommandLine.h, 406
ndkbuild/jni/AIOCommandLine.c, 402
ndkbuild/jni/AIOCommandLine.h, 409
AIOConfiguration
AIOConfiguration.h, 411
ndkbuild/jni/AIOConfiguration.h, 413
AIOConfiguration.c
AIOArgumentInitialize, 410
AIOConfigurationInitialize, 410
AIOConfigurationSetDebug, 410
AIOConfigurationSetTimeout, 410
DeleteAIOConfiguration, 410
NewAIOConfiguration, 410
AIOConfiguration.h
ADCConfiguration, 411
ADCScanType, 412
ADCSetCalFunction, 412
AIOArgumentInitialize, 412
AIOConfiguration, 411
AIOConfigurationInitialize, 412
AIOConfigurationSetDebug, 412
AIOConfigurationSetTimeout, 412
AIOContinousBufConfiguration, 411
ConfigurationType, 412
NewAIOConfiguration, 412
AIOConfigurationInitialize
AIOConfiguration.c, 410
AIOConfiguration.h, 412

ndkbuild/jni/AIOConfiguration.c, 411
ndkbuild/jni/AIOConfiguration.h, 414
AIOConfigurationSetDebug
AIOConfiguration.c, 410
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.c, 411
ndkbuild/jni/AIOConfiguration.h, 414
AIOConfigurationSetTimeout
AIOConfiguration.c, 410
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.c, 411
ndkbuild/jni/AIOConfiguration.h, 414
AIOContinousBufConfiguration
AIOConfiguration.h, 411
ndkbuild/jni/AIOConfiguration.h, 413
AIOContinuousBuf, 159
AIOContinuousBuffer.h, 437
base_size, 160
block_size, 160
buffer, 160
bytes_processed, 160
callback, 160
counter_control, 161
countsbuf, 160
debug, 161
DeviceIndex, 160
exitcode, 161
fifo, 160
hz, 160
lock, 160
mask, 161
ndkbuild/jni/AIOContinuousBuffer.h, 445
num_channels, 160
num_oversamples, 160
num_scans, 160
PopN, 161
PushN, 161
scans_read, 160
size, 160
start_scanning, 160
status, 161
tattr, 160
testing, 161
timeout, 161
type, 161
unit_size, 160
work, 160
worker, 160
AIOContinuousBuf_BufSizeForCounts
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 429
AIOContinuousBuf_GetDeviceIndex
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBuf_GetOverSample
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBuf_InitConfiguration
AIOContinuousBuffer.c, 418
ndkbuild/jni/AIOContinuousBuffer.c, 429
AIOContinuousBuf_NumberChannels
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 431
AIOContinuousBuf_NumberSignals
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 431
AIOContinuousBuf_SaveConfig
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBuf_SendPreConfig
AIOContinuousBuffer.c, 418
ndkbuild/jni/AIOContinuousBuffer.c, 429
AIOContinuousBuf_SetAllGainCodeAndDiffMode
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBuf_SetCallback
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 429
AIOContinuousBuf_SetChannelRange
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBuf_SetChannelRangeGain
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBuf_SetDeviceIndex
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBuf_SetDiscardFirstSample
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBuf_SetOversample
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBuf_SetStartAndEndChannel
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBuf_SetTesting
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBufCallbackStart
AIOContinuousBuffer.c, 423
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufCallbackStartCallbackWithAcquisition-Function
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufCleanup
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 451
AIOContinuousBufCountScansAvailable
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufCountsAvailable
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufEnd
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufForceTerminateAcqusition
AIOContinuousBuffer.c, 418
ndkbuild/jni/AIOContinuousBuffer.c, 428
AIOContinuousBufForceTerminateAcqusitionOverrun
AIOContinuousBuffer.c, 418
ndkbuild/jni/AIOContinuousBuffer.c, 428
AIOContinuousBufGetADCConfigBlock
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429

ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetBaseSize
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetBufferSize
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetCallback
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetClock
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufGetDataAvailable
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufGetDebug
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufGetDeviceIndex
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetExitCode
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetNumberChannels
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 428
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetNumberOfScansToRead
AIOContinuousBuffer.c, 421
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufGetNumberSamplesPerScan
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 439
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufGetNumberScans
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufGetOversample
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetReadPosition
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetRemainingSize
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetRunStatus
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetScansRead
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetSize
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufGetSizeNumElements
AIOContinuousBuffer.c, 419
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufGetStatus
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufGetStreamingBlockSize
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetTesting
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetTimeout
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufGetTotalSamplesExpected
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 439
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufGetUnitSize
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufGetWritePosition
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430

ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufInitADCConfigBlock
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufInitConfiguration
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 445
AIOContinuousBufInitiateCallbackAcquisition
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufLoadCounters
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
AIOContinuousBufLock
AIOContinuousBuffer.c, 423
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufMode
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIOContinuousBufNumberChannels
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufNumberSamplesAvailable
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 439
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufNumberSignals
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufNumberWriteSamplesRemaining
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufPending
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufPopN
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufPreSetup
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
AIOContinuousBufPushN
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufRead
AIOContinuousBuffer.c, 423
AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufReadCompleteScanCounts
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufReadIntegerNumberOfScans
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufReadIntegerScanCounts
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 430
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufReadNSamples
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
AIOContinuousBufReadSingle
AIOContinuousBuffer.c, 421
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufReset
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufResetDevice
AIOContinuousBuffer.c, 423
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSaveConfig
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSendPreConfig
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetAllGainCodeAndDiffMode
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetBaseSize
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetCallback
AIOContinuousBuffer.c, 419

AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetChannelMask
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 447
AIOContinuousBufSetChannelRange
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSetClock
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufSetCountsBuffer
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufSetDebug
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSetDefaultModeForCounterScan
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 433
AIOContinuousBufSetDeviceIndex
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSetDiscardFirstSample
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetNumberChannels
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetNumberOfScansToRead
AIOContinuousBuffer.c, 421
ndkbuild/jni/AIOContinuousBuffer.c, 430
AIOContinuousBufSetNumberScans
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSetOverSample
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
AIOContinuousBufSetOversample
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 438
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetStartAndEndChannel
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetStreamingBlockSize
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetTesting
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetTimeout
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufSetUnitSize
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 439
libaiousb.h, 717
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 446
AIOContinuousBufSetVoltsBuffer
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufSimpleSetupConfig
AIOContinuousBuffer.c, 424
AIOContinuousBuffer.h, 441
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufStart
AIOContinuousBuffer.c, 421
ndkbuild/jni/AIOContinuousBuffer.c, 431
AIOContinuousBufStopAcquisition
AIOContinuousBuffer.c, 421
AIOContinuousBuffer.h, 440
libaiousb.h, 719
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 449
AIOContinuousBufToJSON
AIOContinuousBuffer.c, 425
AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 451
AIOContinuousBufUnlock
AIOContinuousBuffer.c, 423
AIOContinuousBuffer.h, 440
libaiousb.h, 718
ndkbuild/jni/AIOContinuousBuffer.c, 433
ndkbuild/jni/AIOContinuousBuffer.h, 448
AIOContinuousBufWrite
AIOContinuousBuffer.c, 422
AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 431
ndkbuild/jni/AIOContinuousBuffer.h, 450
AIOContinuousBufWriteCounts
AIOContinuousBuffer.c, 422

AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 432
ndkbuild/jni/AIOContinuousBuffer.h, 451
AIOContinuousBuffer.c
AIOContinuousBuf_BufSizeForCounts, 419
AIOContinuousBuf_GetDeviceIndex, 424
AIOContinuousBuf_GetOverSample, 424
AIOContinuousBuf_InitConfiguration, 418
AIOContinuousBuf_NumberChannels, 422
AIOContinuousBuf_NumberSignals, 422
AIOContinuousBuf_SaveConfig, 424
AIOContinuousBuf_SendPreConfig, 418
AIOContinuousBuf_SetAllGainCodeAndDiffMode, 424
AIOContinuousBuf_SetCallback, 419
AIOContinuousBuf_SetChannelRange, 424
AIOContinuousBuf_SetChannelRangeGain, 424
AIOContinuousBuf_SetDeviceIndex, 424
AIOContinuousBuf_SetDiscardFirstSample, 424
AIOContinuousBuf_SetOversample, 424
AIOContinuousBuf_SetStartAndEndChannel, 424
AIOContinuousBuf_SetTesting, 424
AIOContinuousBufCallbackStart, 423
AIOContinuousBufCallbackStartCallbackWith-AcquisitionFunction, 422
AIOContinuousBufCleanup, 422
AIOContinuousBufCountScansAvailable, 419
AIOContinuousBufCountsAvailable, 419
AIOContinuousBufEnd, 424
AIOContinuousBufForceTerminateAcqusition, 418
AIOContinuousBufForceTerminateAcqusitionOverrun, 418
AIOContinuousBufGetADCConfigBlock, 419
AIOContinuousBufGetBaseSize, 418
AIOContinuousBufGetBufferSize, 418
AIOContinuousBufGetCallback, 421
AIOContinuousBufGetClock, 421
AIOContinuousBufGetDataAvailable, 419
AIOContinuousBufGetDebug, 424
AIOContinuousBufGetDeviceIndex, 424
AIOContinuousBufGetExitCode, 419
AIOContinuousBufGetNumberChannels, 418
AIOContinuousBufGetNumberOfScansToRead, 421
AIOContinuousBufGetNumberSamplesPerScan, 422
AIOContinuousBufGetNumberScans, 419
AIOContinuousBufGetOversample, 424
AIOContinuousBufGetReadPosition, 419
AIOContinuousBufGetRemainingSize, 419
AIOContinuousBufGetRunStatus, 419
AIOContinuousBufGetScansRead, 419
AIOContinuousBufGetSize, 419
AIOContinuousBufGetSizeNumElements, 419
AIOContinuousBufGetStatus, 419
AIOContinuousBufGetStreamingBlockSize, 419
AIOContinuousBufGetTesting, 424
AIOContinuousBufGetTimeout, 424
AIOContinuousBufGetTotalSamplesExpected, 422
AIOContinuousBufGetUnitSize, 419
AIOContinuousBufGetWritePosition, 419
AIOContinuousBufInitADCConfigBlock, 418
AIOContinuousBufInitConfiguration, 418
AIOContinuousBufInitiateCallbackAcquisition, 422
AIOContinuousBufLoadCounters, 422
AIOContinuousBufLock, 423
AIOContinuousBufNumberChannels, 422
AIOContinuousBufNumberSamplesAvailable, 422
AIOContinuousBufNumberSignals, 422
AIOContinuousBufNumberWriteSamplesRemaining, 422
AIOContinuousBufPending, 419
AIOContinuousBufPopN, 418
AIOContinuousBufPreSetup, 422
AIOContinuousBufPushN, 418
AIOContinuousBufRead, 423
AIOContinuousBufReadCompleteScanCounts, 421
AIOContinuousBufReadIntegerNumberOfScans, 421
AIOContinuousBufReadIntegerScanCounts, 419
AIOContinuousBufReadNSamples, 422
AIOContinuousBufReadSingle, 421
AIOContinuousBufReset, 419
AIOContinuousBufResetDevice, 423
AIOContinuousBufSaveConfig, 424
AIOContinuousBufSendPreConfig, 419
AIOContinuousBufSetAllGainCodeAndDiffMode, 424
AIOContinuousBufSetBaseSize, 418
AIOContinuousBufSetCallback, 419
AIOContinuousBufSetChannelMask, 421
AIOContinuousBufSetChannelRange, 424
AIOContinuousBufSetClock, 421
AIOContinuousBufSetCountsBuffer, 419
AIOContinuousBufSetDebug, 424
AIOContinuousBufSetDefaultModeForCounterScan, 424
AIOContinuousBufSetDeviceIndex, 424
AIOContinuousBufSetDiscardFirstSample, 424
AIOContinuousBufSetNumberChannels, 418
AIOContinuousBufSetNumberOfScansToRead, 421
AIOContinuousBufSetNumberScans, 419
AIOContinuousBufSetOverSample, 424
AIOContinuousBufSetOversample, 424
AIOContinuousBufSetStartAndEndChannel, 424
AIOContinuousBufSetStreamingBlockSize, 419
AIOContinuousBufSetTesting, 424
AIOContinuousBufSetTimeout, 424
AIOContinuousBufSetUnitSize, 419
AIOContinuousBufSetVoltsBuffer, 419
AIOContinuousBufSimpleSetupConfig, 424
AIOContinuousBufStart, 421
AIOContinuousBufStopAcquisition, 421
AIOContinuousBufToJSON, 425
AIOContinuousBufUnlock, 423
AIOContinuousBufWrite, 422
AIOContinuousBufWriteCounts, 422
BaseSizeRange, 425
continue_running, 422
ConvertCountsToVoltsFunction, 417
DeleteAIOContinuousBuf, 419
GetJSONValueOrDefault, 424
NewAIOContinuousBuf, 418
NewAIOContinuousBufForCounts, 418
NewAIOContinuousBufForVolts, 418
NewAIOContinuousBufFromJSON, 425
number_to_read, 422
RangeValueLookup, 417
RawCountsWorkFunction, 417
ResetCounters, 422
SetConfig, 422
StartStreaming, 422
TrueFalse, 425
AIOContinuousBuffer.h
AIOContinuousBuf, 437
AIOContinuousBufCallbackStart, 440
AIOContinuousBufCallbackStartCallbackWith-AcquisitionFunction, 440
AIOContinuousBufCleanup, 442
AIOContinuousBufCountScansAvailable, 441
AIOContinuousBufEnd, 441
AIOContinuousBufGetADCConfigBlock, 438
AIOContinuousBufGetBaseSize, 438
AIOContinuousBufGetBufferSize, 438
AIOContinuousBufGetCallback, 438
AIOContinuousBufGetClock, 441
AIOContinuousBufGetDebug, 439

AIOContinuousBufGetDeviceIndex, 439
AIOContinuousBufGetExitCode, 441
AIOContinuousBufGetNumberChannels, 438
AIOContinuousBufGetNumberSamplesPerScan, 439
AIOContinuousBufGetNumberScans, 439
AIOContinuousBufGetOversample, 438
AIOContinuousBufGetReadPosition, 441
AIOContinuousBufGetRemainingSize, 441
AIOContinuousBufGetRunStatus, 441
AIOContinuousBufGetScansRead, 441
AIOContinuousBufGetStatus, 441
AIOContinuousBufGetStreamingBlockSize, 438
AIOContinuousBufGetTesting, 439
AIOContinuousBufGetTimeout, 439
AIOContinuousBufGetTotalSamplesExpected, 439
AIOContinuousBufGetUnitSize, 439
AIOContinuousBufGetWritePosition, 441
AIOContinuousBufInitADCConfigBlock, 438
AIOContinuousBufInitConfiguration, 438
AIOContinuousBufInitiateCallbackAcquisition, 441
AIOContinuousBufLock, 440
AIOContinuousBufNumberChannels, 438
AIOContinuousBufNumberSamplesAvailable, 439
AIOContinuousBufNumberSignals, 439
AIOContinuousBufNumberWriteSamplesRemaining, 439
AIOContinuousBufPending, 441
AIOContinuousBufPopN, 440
AIOContinuousBufPushN, 440
AIOContinuousBufRead, 442
AIOContinuousBufReadCompleteScanCounts, 441
AIOContinuousBufReadIntegerNumberOfScans, 441
AIOContinuousBufReadIntegerScanCounts, 441
AIOContinuousBufReset, 439
AIOContinuousBufResetDevice, 439
AIOContinuousBufSaveConfig, 439
AIOContinuousBufSendPreConfig, 439
AIOContinuousBufSetAllGainCodeAndDiffMode, 439
AIOContinuousBufSetBaseSize, 438
AIOContinuousBufSetCallback, 438
AIOContinuousBufSetChannelMask, 439
AIOContinuousBufSetChannelRange, 439
AIOContinuousBufSetClock, 441
AIOContinuousBufSetCountsBuffer, 441
AIOContinuousBufSetDebug, 439
AIOContinuousBufSetDeviceIndex, 439
AIOContinuousBufSetDiscardFirstSample, 439
AIOContinuousBufSetNumberChannels, 438
AIOContinuousBufSetNumberScans, 439
AIOContinuousBufSetOversample, 438
AIOContinuousBufSetStartAndEndChannel, 439
AIOContinuousBufSetStreamingBlockSize, 438
AIOContinuousBufSetTesting, 439
AIOContinuousBufSetTimeout, 439
AIOContinuousBufSetUnitSize, 439
AIOContinuousBufSetVoltsBuffer, 441
AIOContinuousBufSimpleSetupConfig, 441
AIOContinuousBufStopAcquisition, 440
AIOContinuousBufToJSON, 442
AIOContinuousBufUnlock, 440
AIOContinuousBufWrite, 442
AIOContinuousBufWriteCounts, 442
AIOUSB_WorkFn, 437
DeleteAIOContinuousBuf, 438
NewAIOContinuousBuf, 437
NewAIOContinuousBufForCounts, 438
NewAIOContinuousBufForVolts, 438
NewAIOContinuousBufFromJSON, 442
ROOTCLOCK, 437
AIOCountsConverter
AIOCountsConverter.h, 454
libaiousb.h, 708
ndkbuild/jni/AIOCountsConverter.h, 455
AIOCountsConverter.c
AIOCountsConverterConvert, 452
AIOCountsConverterConvertAllAvailableScans, 452
AIOCountsConverterConvertFifo, 452
AIOCountsConverterConvertNScans, 452
AIOCountsConverterReset, 452
Convert, 452
default_out, 451
DeleteAIOCountsConverter, 452
DeleteAIOGainRange, 452
enhanced_out, 451
NewAIOCountsConverter, 451
NewAIOCountsConverterWithBuffer, 451
NewAIOCountsConverterWithScanLimiter, 451
NewAIOGainRangeFromADCConfigBlock, 452
AIOCountsConverter.h
AIOCountsConverter, 454
AIOCountsConverterConvert, 454
AIOCountsConverterConvertAllAvailableScans, 454
AIOCountsConverterConvertFifo, 454
AIOCountsConverterConvertNScans, 454
AIOCountsConverterReset, 454
DeleteAIOCountsConverter, 454
DeleteAIOGainRange, 454
NewAIOCountsConverter, 454
NewAIOCountsConverterFromAIOContinuousBuf, 454
NewAIOCountsConverterWithBuffer, 454
NewAIOCountsConverterWithScanLimiter, 454
NewAIOGainRangeFromADCConfigBlock, 454
AIOCountsConverterConvert
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
AIOCountsConverterConvertAllAvailableScans
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
AIOCountsConverterConvertFifo
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
AIOCountsConverterConvertNScans
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
AIOCountsConverterReset
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
AIODeviceInfo, 161
AIODeviceInfo.h, 457
Counters, 161
DIOBytes, 161
Name, 161
NameSize, 161
ndkbuild/jni/AIODeviceInfo.h, 458
PID, 161
AIODeviceInfo.c
AIODeviceInfoGet, 456
AIODeviceInfoGetCounters, 456

AIODeviceInfoGetDIOBytes, 456
AIODeviceInfoGetName, 456
DeleteAIODeviceInfo, 456
NewAIODeviceInfo, 456
AIODeviceInfo.h
AIODeviceInfo, 457
AIODeviceInfoGet, 457
AIODeviceInfoGetCounters, 457
AIODeviceInfoGetDIOBytes, 457
AIODeviceInfoGetName, 457
DeleteAIODeviceInfo, 457
NewAIODeviceInfo, 457
AIODeviceInfoGet
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
AIODeviceInfoGetCounters
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
AIODeviceInfoGetDIOBytes
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
AIODeviceInfoGetName
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
AIODeviceQuery, 161
AIODeviceQuery.h, 465
index, 162
name, 162
nameSize, 162
ndkbuild/jni/AIODeviceQuery.h, 468
numCounters, 162
numDIOBytes, 162
productID, 162
AIODeviceQuery.c
AIODeviceQueryGetIndex, 460
AIODeviceQueryGetName, 461
AIODeviceQueryGetNumCounters, 461
AIODeviceQueryGetNumDIOBytes, 461
AIODeviceQueryGetProductID, 460
AIODeviceQueryNameSize, 460
AIODeviceQueryToRepr, 460
AIODeviceQueryToStr, 460
DeleteAIODeviceQuery, 460
NewAIODeviceQuery, 459
AIODeviceQuery.h
AIODeviceQuery, 465
AIODeviceQueryGetIndex, 467
AIODeviceQueryGetName, 466
AIODeviceQueryGetNumCounters, 466
AIODeviceQueryGetNumDIOBytes, 466
AIODeviceQueryGetProductID, 466
AIODeviceQueryNameSize, 466
AIODeviceQueryToRepr, 466
AIODeviceQueryToStr, 465
DeleteAIODeviceQuery, 465
NewAIODeviceQuery, 465
AIODeviceQueryGetIndex
AIODeviceQuery.c, 460
AIODeviceQuery.h, 467
libaiousb.h, 728
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 469
AIODeviceQueryGetName
AIODeviceQuery.c, 461
AIODeviceQuery.h, 466
libaiousb.h, 728
ndkbuild/jni/AIODeviceQuery.c, 464
ndkbuild/jni/AIODeviceQuery.h, 469
AIODeviceQueryGetNumCounters
AIODeviceQuery.c, 461
AIODeviceQuery.h, 466
libaiousb.h, 728
ndkbuild/jni/AIODeviceQuery.c, 464
ndkbuild/jni/AIODeviceQuery.h, 469
AIODeviceQueryGetNumDIOBytes
AIODeviceQuery.c, 461
AIODeviceQuery.h, 466
libaiousb.h, 728
ndkbuild/jni/AIODeviceQuery.c, 464
ndkbuild/jni/AIODeviceQuery.h, 469
AIODeviceQueryGetProductID
AIODeviceQuery.c, 460
AIODeviceQuery.h, 466
libaiousb.h, 727
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 468
AIODeviceQueryNameSize
AIODeviceQuery.c, 460
AIODeviceQuery.h, 466
libaiousb.h, 728
ndkbuild/jni/AIODeviceQuery.c, 464
ndkbuild/jni/AIODeviceQuery.h, 468
AIODeviceQueryToRepr
AIODeviceQuery.c, 460
AIODeviceQuery.h, 466
libaiousb.h, 727
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 468
AIODeviceQueryToStr
AIODeviceQuery.c, 460
AIODeviceQuery.h, 465
libaiousb.h, 727
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 468
AIODeviceTable.c
AIODeviceTableAddDeviceToDeviceTable, 472
AIODeviceTableAddDeviceToDeviceTableWithUSB-Device, 472
AIODeviceTableClearDevices, 472
AIODeviceTableGetAIOUSBDeviceAtIndex, 472
AIODeviceTableGetDeviceAtIndex, 472
AIODeviceTableGetUSBDeviceAtIndex, 471
AIODeviceTableInit, 470
AIODeviceTablePopulateTable, 472
AIODeviceTablePopulateTableTest, 472
AIODeviceTableSetDeviceID, 472
AIOUSB_Cleanup, 470
AIOUSB_EnsureOpen, 472
AIOUSB_Exit, 473
AIOUSB_GetAllDevices, 472
AIOUSB_Init, 473
AIOUSB_InitTest, 470
AIOUSB_IsInit, 470
AIOUSB_Reset, 473
AIOUSB_SetInit, 470
AIOUSBGetError, 472
aiousbInit, 473
ClearAIODeviceTable, 472
ClearDevices, 472
CloseAllDevices, 472
deviceTable, 473
GetDevices, 471

GetSafeDeviceName, 472
NUM_PROD_NAMES, 470
ProductIDToName, 471
ProductNameToID, 471
QueryDeviceInfo, 470
AIODeviceTable.h
AIODeviceTableAddDeviceToDeviceTable, 477
AIODeviceTableAddDeviceToDeviceTableWithUSB-Device, 477
AIODeviceTableClearDevices, 478
AIODeviceTableGetAIOUSBDeviceAtIndex, 478
AIODeviceTableGetDeviceAtIndex, 478
AIODeviceTableGetUSBDeviceAtIndex, 478
AIODeviceTableInit, 480
AIODeviceTablePopulateTable, 478
AIODeviceTablePopulateTableTest, 478
AIOUSB_EnsureOpen, 479
AIOUSB_Exit, 480
AIOUSB_GetAllDevices, 480
AIOUSB_Init, 479
AIOUSB_IsInit, 480
AIOUSB_Reset, 480
AIOUSBGetError, 480
ClearAIODeviceTable, 480
ClearDevices, 478
CloseAllDevices, 480
deviceTable, 480
GetDevices, 478
GetSafeDeviceName, 479
ProductIDToName, 479
ProductNameToID, 479
QueryDeviceInfo, 478
AIODeviceTableAddDeviceToDeviceTable
AIODeviceTable.c, 472
AIODeviceTable.h, 477
libaiousb.h, 728
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice
AIODeviceTable.c, 472
AIODeviceTable.h, 477
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableClearDevices
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableGetAIOUSBDeviceAtIndex
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableGetDeviceAtIndex
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 475
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableGetUSBDeviceAtIndex
AIODeviceTable.c, 471
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 475
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableInit
AIODeviceTable.c, 470
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIODeviceTable.h, 483
AIODeviceTablePopulateTable
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTablePopulateTableTest
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
AIODeviceTableSetDeviceID
AIODeviceTable.c, 472
ndkbuild/jni/AIODeviceTable.c, 476
AIODisplayType
AIOUSB_Properties.h, 655
ndkbuild/jni/AIOUSB_Properties.h, 656
AIOEither
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
AIOEither.c
AIOEitherClear, 485
AIOEitherGetLeft, 485
AIOEitherGetRight, 485
AIOEitherHasError, 485
AIOEitherSetLeft, 485
AIOEitherSetRight, 485
AIOEitherToAIONumber, 485
AIOEitherToAIORetType, 485
AIOEitherToDouble, 485
AIOEitherToInt, 485
AIOEitherToShort, 485
AIOEitherToString, 485
AIOEitherToUnsigned, 485
LOOKUP, 484
AIOEither.h
AIO_EITHER_TYPE, 487
AIO_ERROR_VALUE, 487
AIOEither, 487
AIOEitherClear, 488
AIOEitherGetLeft, 488
AIOEitherGetRight, 488
AIOEitherHasError, 488
AIOEitherSetLeft, 488
AIOEitherSetRight, 488
AIOEitherToAIONumber, 488
AIOEitherToAIORetType, 488
AIOEitherToDouble, 488
AIOEitherToInt, 488
AIOEitherToShort, 488
AIOEitherToString, 488
AIOEitherToUnsigned, 488
AIOEitherClear
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherGetLeft
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherGetRight
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486

ndkbuild/jni/AIOEither.h, 490
AIOEitherHasError
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherSetLeft
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherSetRight
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToAIONumber
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToAIORetType
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToDouble
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToInt
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToShort
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToString
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOEitherToUnsigned
AIOEither.c, 485
AIOEither.h, 488
libaiousb.h, 732
ndkbuild/jni/AIOEither.c, 486
ndkbuild/jni/AIOEither.h, 490
AIOError
AIOUSB_Core.h, 616
ndkbuild/jni/AIOUSB_Core.h, 620
AIOFifo, 163
AIOFifo.h, 495
LOCKING_MECHANISM, 163
ndkbuild/jni/AIOFifo.h, 498
AIOFifo.c
AIOFifoAllOrNoneInitialize, 491
AIOFifoGetRefSize, 491
AIOFifoGetSize, 491
AIOFifoGetSizeNumElements, 491
AIOFifoInitialize, 491
AIOFifoRead, 491
AIOFifoReadAllOrNone, 492
AIOFifoReadPosition, 492
AIOFifoReadSize, 491
AIOFifoReadSizeNumElements, 491
AIOFifoReset, 491
AIOFifoResize, 491
AIOFifoWrite, 491
AIOFifoWriteAllOrNone, 491
AIOFifoWritePosition, 492
AIOFifoWriteSizeRemaining, 491
AIOFifoWriteSizeRemainingNumElements, 491
DeleteAIOFifo, 491
DeleteAIOFifoTYPE, 491
delta, 491
increment, 491
LOOKUP, 491
NewAIOFifo, 491
NewAIOFifoAllOrNone, 491
NewAIOFifoTYPE, 491
Pop, 491
PopN, 491
Push, 491
PushN, 491
rdelta, 491
AIOFifo.h
AIOFifo, 495
AIOFifoGetRefSize, 496
AIOFifoGetSize, 496
AIOFifoGetSizeNumElements, 496
AIOFifoRead, 496
AIOFifoReadAllOrNone, 496
AIOFifoReadPosition, 496
AIOFifoReadSize, 496
AIOFifoReadSizeNumElements, 496
AIOFifoReset, 496
AIOFifoResize, 496
AIOFifoTYPE, 495
AIOFifoWrite, 496
AIOFifoWriteAllOrNone, 496
AIOFifoWritePosition, 496
AIOFifoWriteSizeRemaining, 496
AIOFifoWriteSizeRemainingNumElements, 496
DeleteAIOFifo, 496
GRAB_RESOURCE, 495
INPUT_TYPE, 495
LOCKING_MECHANISM, 495
NewAIOFifo, 496
NewAIOFifoTYPE, 496
PopN, 496
Push, 496
PushN, 496
RELEASE_RESOURCE, 495
TYPE, 495
AIOFifoAllOrNoneInitialize
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
AIOFifoGetRefSize
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoGetSize
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoGetSizeNumElements

AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoInitialize
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
AIOFifoRead
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoReadAllOrNone
AIOFifo.c, 492
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoReadPosition
AIOFifo.c, 492
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoReadSize
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoReadSizeNumElements
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoReset
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoResize
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoTYPE
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 498
AIOFifoWrite
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoWriteAllOrNone
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
AIOFifoWritePosition
AIOFifo.c, 492
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoWriteSizeRemaining
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOFifoWriteSizeRemainingNumElements
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 732
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
AIOGainRange, 163
max, 163
min, 163
AIOGetChannelRange
AIOCommandLine.c, 398
AIOCommandLine.h, 405
libaiousb.h, 747
ndkbuild/jni/AIOCommandLine.c, 402
ndkbuild/jni/AIOCommandLine.h, 409
AIOList.c
Deleteint, 499
Deleteintlist, 500
intToString, 499
intlistFirst, 499
intlistInsert, 500
intlistSize, 500
intlistToString, 500
Newintlist, 499
newone, 500
AIOList.h
Deleteintlist, 502
foreach_CStringArray_p, 502
foreach_int, 502
intlistFirst, 502
intlistInsert, 502
intlistSize, 502
intlistToString, 502
Newintlist, 502
TAIL_Q_LIST, 501, 502
AIOOption
AIOUSB_Core.h, 616
ndkbuild/jni/AIOUSB_Core.h, 620
AIOPrintUsage
AIOCommandLine.c, 397
AIOCommandLine.h, 405
libaiousb.h, 747
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 409
AIOProcessCmdline
AIOCommandLine.c, 397
AIOCommandLine.h, 404
libaiousb.h, 746
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 408
AIOProcessCommandLine
AIOCommandLine.c, 397
AIOCommandLine.h, 405
libaiousb.h, 746
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 408
AIOProductGroup, 163
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.h, 509
AIOProductGroupContains
AIOProductTypes.c, 505
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
AIOProductRange, 164
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.h, 509

AIOProductRangeEnd
AIOProductTypes.c, 505
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
AIOProductRangeStart
AIOProductTypes.c, 504
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
AIOProductTypes.c
AIOProductGroupContains, 505
AIOProductRangeEnd, 505
AIOProductRangeStart, 504
DeleteAIOProductGroup, 505
DeleteAIOProductRange, 504
groupcpy, 505
NewAIOProductGroup, 505
NewAIOProductRange, 504
AIOProductTypes.h
AIO_RANGE, 507
AIOProductGroup, 507
AIOProductGroupContains, 507
AIOProductRange, 507
AIOProductRangeEnd, 507
AIOProductRangeStart, 507
DeleteAIOProductGroup, 507
DeleteAIOProductRange, 507
groupcpy, 507
NUMARGS, 507
NewAIOProductGroup, 507
NewAIOProductRange, 507
AIORESULT
AIOTypes.h, 520
libaiousb.h, 708
ndkbuild/jni/AIOTypes.h, 542
AIORET_TYPE
AIOTypes.h, 520
libaiousb.h, 708
ndkbuild/jni/AIOTypes.h, 542
AIOTUPLE2
AIOTuple.h, 511
ndkbuild/jni/AIOTuple.h, 511
AIOTUPLE2_PTR
AIOTuple.h, 511
ndkbuild/jni/AIOTuple.h, 511
AIOTUPLE2_TO_STR
AIOTuple.h, 511
ndkbuild/jni/AIOTuple.h, 511
AIOTUPLE2_TYPE
AIOTuple.h, 510, 511
ndkbuild/jni/AIOTuple.h, 511
AIOTuple.h
AIO_CHAR_ARRAY, 510
AIOTUPLE2, 511
AIOTUPLE2_PTR, 511
AIOTUPLE2_TO_STR, 511
AIOTUPLE2_TYPE, 510, 511
AIOTypes.h
ACCES_DEPRECATED, 518
ADCalMode, 533
ADGainCode, 530
ADRegister, 529
AIO_ASSERT, 518
AIO_ASSERT_EXIT, 518
AIO_ASSERT_RET, 518
AIO_ASSERT_USB, 519
AIO_ERROR, 519
AIO_MAKE_ERROR, 518
AIO_NUMBER, 520
AIO_SCAN_TYPE, 520
AIOBufferType, 520
AIOCommandCode, 533
AIOContinuousBufMode, 520
AIORESULT, 520
AIORET_TYPE, 520
AIOUSB_BOOL, 520
AIOUSB_BOOL_VAL, 521
AUR_CBUF_EXIT, 518
AUR_CBUF_SETUP, 518
COUNTS, 520
CREATE_ENUM, 517
DACRange, 525
EXPORTED_FUNCTION, 517
EnumStringLookup, 520
FIFO_Method, 526
FIRST_ENUM, 518
foreach_array, 518
G_BREAKPOINT, 519
G_STMT_END, 519
G_STMT_START, 519
GCC_VERSION, 518
HAS_PTHREAD, 517
LAMBDA, 518
LAST_ENUM, 517
MAX, 518
MAX_VALUE, 518
MIN, 518
MIN_VALUE, 518
NUMBER_CHANNELS, 518
ProductIDS, 521
ROOTCLOCK, 519
ResultCode, 526
THREAD_STATUS, 520
Ushort_Array, 520
VALID_ENUM, 518
VALID_PRODUCT, 518
VENDOR_REQUEST, 530
AIOUSB, 143
operator<<, 144
AIOUSB::AI16_DataPoint
AI16_DataPoint, 147
AI16_DataPointArray, 148
AI16_DataSet, 148
AnalogInputSubsystem, 148
channel, 148
counts, 148
differentialMode, 148
getChannel, 147
getCounts, 147
getRange, 147
getRangeText, 147
getVolts, 148
isDifferentialMode, 147
range, 148
std::vector< AI16_DataPoint >, 148
toString, 148
AIOUSB::AI16_DataPointArray
AI16_DataPointArray, 148
AIOUSB::AI16_DataSet
~AI16_DataSet, 149
AI16_DataSet, 149
AnalogInputSubsystem, 151
calMode, 151
discardFirstSample, 151
getCalMode, 150
getOverSample, 150
getPoints, 150
getSubsystem, 150
getTimeStamp, 150
getTriggerMode, 150
isDiscardFirstSample, 150
overSample, 151
points, 151

print, 151
subsystem, 151
timeStamp, 151
triggerMode, 151
AIOUSB::AI16_InputRange
~AI16_InputRange, 152
AI16_DataPoint, 152
AI16_InputRange, 152
AnalogInputSubsystem, 152
setRange, 152
AIOUSB::AO16_AnalogOutputSubsystem
AO16_AnalogOutputSubsystem, 193
countsToVolts, 194
getRange, 193
getRangeText, 193
outputRange, 195
RANGE_10V, 195
RANGE_5V, 195
setRange, 194
voltsToCounts, 194
writeVolts, 194
AIOUSB::AO16_OutputRange
~AO16_OutputRange, 196
AO16_AnalogOutputSubsystem, 196
AO16_OutputRange, 196
setRange, 196
AIOUSB::AnalogIORange
~AnalogIORange, 189
AnalogIORange, 188
countsToVolts, 189
getRange, 189
maxCounts, 190
maxVolts, 190
minCounts, 190
minVolts, 190
range, 190
rangeCounts, 190
rangeVolts, 190
setCountRange, 189
setRange, 189
setVoltRange, 189
voltsToCounts, 190
AIOUSB::AnalogInputSubsystem
~AnalogInputSubsystem, 172
AnalogInputSubsystem, 172
autoCalFeature, 187
autoConfig, 187
calMode, 187
calibrate, 180, 181
channelsPerGroup, 187
clearFIFO, 183
configBlockSize, 187
countsToVolts, 184
differentialMode, 187
endChannel, 187
getCalMode, 175
getChannelsPerGroup, 172
getClock, 180
getNumChannels, 172
getNumMUXChannels, 172
getOverSample, 179
getRange, 176
getRangeText, 173
getStreamingBlockSize, 180
getTriggerMode, 175
inputRange, 187
isAutoCalPresent, 172
isAutoConfig, 173
isDifferentialMode, 177
isDiscardFirstSample, 174
MAX_CHANNELS, 187
MAX_COUNTS, 186
MIN_COUNTS, 186
numChannels, 187
numMUXChannels, 187
overSample, 187
print, 172
RANGE_0_10V, 186
RANGE_0_1V, 186
RANGE_0_2V, 186
RANGE_0_5V, 186
RANGE_10V, 186
RANGE_1V, 186
RANGE_2V, 186
RANGE_5V, 186
RANGE_TEXT, 186
read, 181
readBulkBuffer, 187
readBulkNext, 183
readBulkSamplesAvailable, 183
readBulkSamplesRequested, 187
readBulkSamplesRetrieved, 187
readBulkStart, 182
readConfig, 174
readCounts, 182
readVolts, 182
setAutoConfig, 173
setCalMode, 175
setCalibrationTable, 179
setClock, 180
setDifferentialMode, 177, 178
setDiscardFirstSample, 174
setOverSample, 179
setRange, 176
setRangeAndDiffMode, 178, 179
setScanRange, 172
setStreamingBlockSize, 180
setTriggerMode, 175
startChannel, 187
triggerMode, 187
USB_AI16_Family, 185
voltsToCounts, 184, 185
writeConfig, 174
AIOUSB::AnalogOutputSubsystem
~AnalogOutputSubsystem, 191
AnalogOutputSubsystem, 191
getNumChannels, 191
maxCounts, 192
minCounts, 192
numChannels, 192
print, 191
writeCounts, 191
AIOUSB::BoolArray
BoolArray, 196
AIOUSB::Counter
Counter, 201
counterIndex, 203
CounterSubsystem, 203
getDeviceIndex, 201
MODE_ONE_SHOT, 203
parent, 203
readCount, 202
readCountAndSetModeAndCount, 203
readCountAndStatus, 202
setCount, 202
setMode, 201
setModeAndCount, 202
AIOUSB::CounterSubsystem
~CounterSubsystem, 205
Counter, 207
CounterSubsystem, 205
counters, 207
getCounter, 205
getNumCounterBlocks, 205

getNumCounters, 205
numCounterBlocks, 207
numCounters, 207
print, 205
readCounts, 205
selectGate, 206
startClock, 206
stopClock, 206
USB_AI16_Family, 207
USB_CTR_15_Family, 207
USB_DIO_32_Family, 207
AIOUSB::DA12_AnalogOutputSubsystem
countsToVolts, 211
DA12_AnalogOutputSubsystem, 208
getRange, 209
getRangeText, 208
outputRange, 212
RANGE_10V, 212
RANGE_5V, 211
setRange, 209, 210
voltsToCounts, 211
writeVolts, 210
AIOUSB::DA12_OutputRange
~DA12_OutputRange, 213
DA12_AnalogOutputSubsystem, 213
DA12_OutputRange, 213
setRange, 213
AIOUSB::DIOStreamSubsystem
~DIOStreamSubsystem, 226
clearFIFO, 228
clockHz, 228
close, 227
DIOStreamSubsystem, 226
getClock, 226
getStreamingBlockSize, 226
open, 227
print, 226
read, 228
setClock, 227
setStreamingBlockSize, 226
stopClock, 227
write, 228
AIOUSB::DeviceSubsystem
~DeviceSubsystem, 218
DeviceSubsystem, 218
getDeviceIndex, 218
getParent, 219
parent, 219
print, 218
AIOUSB::DigitalIOSubsystem
~DigitalIOSubsystem, 220
bitsToBytes, 220
bytesToBits, 220
configure, 221, 222
DigitalIOSubsystem, 220
getConfiguration, 222
getNumChannels, 221
getNumPorts, 221
getNumTristateChannels, 221
getNumTristateGroups, 221
numChannels, 224
numPorts, 224
numTristateChannels, 224
numTristateGroups, 224
print, 220
read, 222, 223
USB_AI16_Family, 223
USB_AO16_Family, 223
USB_DIO_Family, 223
write, 223
writeValues, 224
AIOUSB::DoubleArray
DoubleArray, 229
AIOUSB::IllegalArgumentException
IllegalArgumentException, 234
AIOUSB::IntArray
IntArray, 234
AIOUSB::OperationFailedException
OperationFailedException, 262
AIOUSB::OutputVoltagePoint
channel, 265
OutputVoltagePoint, 265
volts, 265
AIOUSB::OutputVoltagePointArray
OutputVoltagePointArray, 266
AIOUSB::StringArray
StringArray, 268
AIOUSB::UCharArray
UCharArray, 276
AIOUSB::USB_AI16_Family
adc, 280
analogInputSubsystem, 280
counterSubsystem, 280
ctr, 280
digitalIOSubsystem, 280
dio, 280
getSupportedProductIDs, 279
getSupportedProductNames, 279
initialize, 279
isSupportedProductID, 279
print, 279
supportedProductIDs, 280
USBDeviceManager, 280
AIOUSB::USB_AIO16_Family
adc, 282
analogInputSubsystem, 283
analogOutputSubsytem, 283
counterSubsystem, 283
ctr, 283
dac, 282
digitalIOSubsystem, 283
dio, 283
getSupportedProductNames, 282
initialize, 282
print, 282
supportedProductIDs, 283
AIOUSB::USB_AO16_Family
analogOutputSubsystem, 285
dac, 285
digitalIOSubsystem, 285
dio, 285
getSupportedProductIDs, 284
getSupportedProductNames, 284
initialize, 284
isSupportedProductID, 285
print, 285
supportedProductIDs, 285
USBDeviceManager, 285
AIOUSB::USB_CTR_15_Family
counterSubsystem, 289
ctr, 289
initialize, 288
print, 288
AIOUSB::USB_DA12_8A_Family
dac, 291
initialize, 290
print, 291
AIOUSB::USB_DA12_8E_Family
dac, 294
initialize, 292
print, 294
AIOUSB::USB_DIO_16_Family
dio, 297
diostream, 297

initialize, 296
print, 297
AIOUSB::USB_DIO_32_Family
counterSubsystem, 300
ctr, 299
dio, 299
initialize, 299
print, 299
AIOUSB::USB_DIO_Family
digitalIOSubsystem, 302
dio, 302
getSupportedProductNames, 301
initialize, 301
print, 301
supportedProductIDs, 302
USBDeviceManager, 302
AIOUSB::USBDeviceArray
USBDeviceArray, 314
AIOUSB::USBDeviceBase
~USBDeviceBase, 315
AnalogInputSubsystem, 318
clearFIFO, 315
customEEPROMRead, 317
customEEPROMWrite, 317
DIOStreamSubsystem, 318
deviceIndex, 318
getCommTimeout, 316
getDeviceIndex, 316
getMiscClock, 315
getName, 316
getProductID, 316
getSerialNumber, 316
getStreamingBlockSize, 316
name, 318
print, 316
productID, 318
reset, 317
serialNumber, 318
setCommTimeout, 316
setMiscClock, 315
setStreamingBlockSize, 316
USBDeviceBase, 315
USBDeviceManager, 318
AIOUSB::USBDeviceManager
~USBDeviceManager, 321
close, 324
deviceList, 325
emptyDeviceList, 321
getAIOUSBVersion, 321
getAIOUSBVersionDate, 321
getDeviceByProductID, 324, 325
getDeviceBySerialNumber, 325
getResultCodeAsString, 323
isOpen, 323
listDevices, 323
open, 323
openStatus, 325
print, 321
printDevices, 321
productIDToName, 321, 322
productNameToID, 322, 323
scanForDevices, 324
USBDeviceManager, 321
AIOUSB::UShortArray
UShortArray, 335
AIOUSB_ADC.c
ADC_ADMode, 562
ADC_BulkAcquire, 564
ADC_BulkPoll, 565
ADC_CanCalibrate, 564
ADC_ClearADBuf, 565
ADC_ClearFastITConfig, 565
ADC_CopyConfig, 562
ADC_CreateADBuf, 565
ADC_CreateFastITConfig, 565
ADC_GetChannelV, 560
ADC_GetConfig, 561
ADC_GetConfigRegisters, 559
ADC_GetConfigSize, 559
ADC_GetFastITScanV, 566
ADC_GetITScanV, 566
ADC_GetMaxClockRate, 563
ADC_GetOversample, 563
ADC_GetScan, 561
ADC_GetScanV, 561
ADC_InitFastITScanV, 565
ADC_Initialize, 564
ADC_QueryCal, 564
ADC_Range1, 562
ADC_RangeAll, 562
ADC_ReadADConfigBlock, 559
ADC_ResetDevice, 559
ADC_ResetFastITScanV, 565
ADC_SetCal, 563
ADC_SetConfig, 562
ADC_SetOversample, 563
ADC_SetScanLimits, 563
ADConfigBlockToYAML, 565
AIOUSB_GetCalMode, 568
AIOUSB_GetEndChannel, 568
AIOUSB_GetGainCode, 567
AIOUSB_GetOversample, 568
AIOUSB_GetRegister, 567
AIOUSB_GetScan, 559
AIOUSB_GetStartChannel, 568
AIOUSB_GetTriggerMode, 568
AIOUSB_IsDifferentialMode, 567
AIOUSB_SetCalMode, 568
AIOUSB_SetDifferentialMode, 568
AIOUSB_SetGainCode, 567
AIOUSB_SetOversample, 568
AIOUSB_SetRangeSingle, 567
AIOUSB_SetRegister, 567
AIOUSB_SetScanRange, 568
AIOUSB_SetTriggerMode, 568
AIOUSB_VoltsToCounts, 560
adRanges, 569
adc_get_bulk_data, 559
adcblock_valid_channel_settings, 562
adcblock_valid_size, 562
adcblock_valid_trigger_settings, 561
ConfigureAndBulkAcquire, 567
CreateSmartBuffer, 564
dRef, 569
DoLoadCalTable, 567
GetHiRef, 566
ReadConfigBlock, 559
valid_config_block, 562
WriteConfigBlock, 559
AIOUSB_ADC.h
ADC_ADMode, 590
ADC_BulkAcquire, 591
ADC_BulkPoll, 591
ADC_CanCalibrate, 591
ADC_CreateFastITConfig, 591
ADC_GetChannelV, 586
ADC_GetConfig, 589
ADC_GetFastITScanV, 592
ADC_GetITScanV, 592
ADC_GetMaxClockRate, 590
ADC_GetOversample, 592
ADC_GetScan, 588
ADC_GetScanV, 586
ADC_InitFastITScanV, 591

ADC_Initialize, 591
ADC_QueryCal, 590
ADC_Range1, 589
ADC_RangeAll, 586
ADC_ResetFastITScanV, 592
ADC_SetCal, 590
ADC_SetConfig, 589
ADC_SetOversample, 592
ADC_SetScanLimits, 590
AIOUSB_GetCalMode, 596
AIOUSB_GetEndChannel, 596
AIOUSB_GetGainCode, 593
AIOUSB_GetOversample, 596
AIOUSB_GetRegister, 598
AIOUSB_GetScan, 588
AIOUSB_GetStartChannel, 596
AIOUSB_GetTriggerMode, 596
AIOUSB_IsDifferentialMode, 593
AIOUSB_SetCalMode, 596
AIOUSB_SetDifferentialMode, 594
AIOUSB_SetGainCode, 593
AIOUSB_SetOversample, 596
AIOUSB_SetRegister, 598
AIOUSB_SetScanRange, 596
AIOUSB_SetTriggerMode, 596
CreateSmartBuffer, 596
ReadConfigBlock, 592
WriteConfigBlock, 592
AIOUSB_ADC_ExternalCal
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 593
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 606
AIOUSB_ADC_InternalCal
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 598
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 581
ndkbuild/jni/AIOUSB_ADC.h, 609
AIOUSB_ADC_LoadCalTable
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_ADC_SetCalTable
AIOUSB_Core.c, 611
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_ArrayCountsToVolts
AIOUSB_ADC.c, 560
ndkbuild/jni/AIOUSB_ADC.c, 573
AIOUSB_ArrayVoltsToCounts
AIOUSB_ADC.c, 560
ndkbuild/jni/AIOUSB_ADC.c, 573
AIOUSB_BOOL
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
AIOUSB_BOOL_VAL
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
AIOUSB_CTR.c
CTR_8254Load, 624
CTR_8254Mode, 624
CTR_8254ModeLoad, 624
CTR_8254Read, 624
CTR_8254ReadAll, 625
CTR_8254ReadLatched, 625
CTR_8254ReadModeLoad, 624
CTR_8254ReadStatus, 625
CTR_8254SelectGate, 625
CTR_CalculateCountersForClock, 625
CTR_StartOutputFreq, 625
AIOUSB_CTR.h
CTR_8254Load, 627
CTR_8254Mode, 627
CTR_8254ModeLoad, 627
CTR_8254Read, 628
CTR_8254ReadAll, 628
CTR_8254ReadLatched, 628
CTR_8254ReadModeLoad, 628
CTR_8254ReadStatus, 628
CTR_8254SelectGate, 628
CTR_CalculateCountersForClock, 627
CTR_StartOutputFreq, 628
AIOUSB_Cleanup
AIODeviceTable.c, 470
ndkbuild/jni/AIODeviceTable.c, 474
AIOUSB_ClearFIFO
AIOUSB_Core.c, 611
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_Copy_Config_Block
AIOUSB_ADC.c, 566
ndkbuild/jni/AIOUSB_ADC.c, 581
AIOUSB_Core.c
AIOUSB_ClearFIFO, 611
AIOUSB_GetCommTimeout, 611
AIOUSB_GetConfigBlock, 611
AIOUSB_GetDevice, 611
AIOUSB_GetMiscClock, 611
AIOUSB_GetStreamingBlockSize, 611
AIOUSB_GetVersion, 611
AIOUSB_GetVersionDate, 611
AIOUSB_InitConfigBlock, 611
AIOUSB_Lock, 610
AIOUSB_ResetChip, 610
AIOUSB_SetCommTimeout, 611
AIOUSB_SetMiscClock, 611
AIOUSB_SetStreamingBlockSize, 611
AIOUSB_UnLock, 610
AIOUSB_Validate, 610
AIOUSB_Validate_Device, 611
AIOUSB_Validate_Lock, 610
aio_errno, 612
DeviceTableAtIndex, 610
DeviceTableAtIndex_Lock, 611
GenericVendorRead, 612
GenericVendorWrite, 612
productIDNameTable, 612
ResolveDeviceIndex, 610
AIOUSB_Core.h
ADC_ResetDevice, 616
AIOError, 616
AIOOption, 616
AIOUSB_ClearFIFO, 618
AIOUSB_GetCommTimeout, 617
AIOUSB_GetConfigBlock, 617
AIOUSB_GetDevice, 617
AIOUSB_GetDeviceSerialNumber, 616
AIOUSB_GetMiscClock, 617
AIOUSB_GetStreamingBlockSize, 618
AIOUSB_GetVersion, 617
AIOUSB_GetVersionDate, 617
AIOUSB_InitConfigBlock, 618
AIOUSB_InitTest, 617
AIOUSB_Lock, 616
AIOUSB_ResetChip, 616
AIOUSB_SetCommTimeout, 617

AIOUSB_SetMiscClock, 617
AIOUSB_UnLock, 617
AIOUSB_Validate, 617
AIOUSB_Validate_Device, 618
AIOUSB_Validate_Lock, 617
AIOUSB_VoltsToCounts, 617
adRanges, 618
aio_errno, 618
aiousbInit, 618
DeviceTableAtIndex, 617
DeviceTableAtIndex_Lock, 617
GenericVendorRead, 618
GenericVendorWrite, 618
PRIVATE, 616
AIOUSB_DAC.c
DACDirect, 632
DACMultiDirect, 632
DACOutputClose, 633
DACOutputCloseNoEnd, 633
DACOutputFrame, 633
DACOutputFrameRaw, 633
DACOutputOpen, 633
DACOutputSetCount, 633
DACOutputSetInterlock, 633
DACOutputStart, 633
DACSetBoardRange, 633
AIOUSB_DAC.h
DACDirect, 635
DACMultiDirect, 635
DACOutputClose, 636
DACOutputCloseNoEnd, 636
DACOutputFrame, 636
DACOutputFrameRaw, 636
DACOutputOpen, 636
DACOutputSetCount, 636
DACOutputSetInterlock, 636
DACOutputStart, 636
DACSetBoardRange, 636
AIOUSB_DEBUG
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_DEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_DIO.c
aiousb_htons, 639
DIO_ConfigurationQuery, 639
DIO_Configure, 639
DIO_ConfigureEx, 639
DIO_ConfigureWithDIOBuf, 639
DIO_Read1, 639
DIO_Read8, 639
DIO_ReadAll, 639
DIO_ReadAllToCharStr, 639
DIO_ReadIntoDIOBuf, 639
DIO_StreamClose, 639
DIO_StreamFrame, 639
DIO_StreamOpen, 639
DIO_StreamSetClocks, 639
DIO_Write1, 639
DIO_Write8, 639
DIO_WriteAll, 639
pow_of_minsize, 639
AIOUSB_DIO.h
DIO_ConfigurationQuery, 642
DIO_Configure, 642
DIO_ConfigureEx, 642
DIO_ConfigureWithDIOBuf, 642
DIO_Read1, 644
DIO_Read8, 644
DIO_ReadAll, 644
DIO_ReadAllToCharStr, 644
DIO_ReadIntoDIOBuf, 642
DIO_StreamClose, 644
DIO_StreamFrame, 644
DIO_StreamOpen, 644
DIO_StreamSetClocks, 644
DIO_Write1, 642
DIO_Write8, 642
DIO_WriteAll, 642
AIOUSB_ERROR
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_ERROR_VALUE
AIOTypes.h, 518
AIOUSB_EnsureOpen
AIODeviceTable.c, 472
AIODeviceTable.h, 479
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 475
ndkbuild/jni/AIODeviceTable.h, 483
AIOUSB_Exit
AIODeviceTable.c, 473
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 483
AIOUSB_FATAL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_FindDeviceIndicesByGroup
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 656
libaiousb.h, 745
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 658
AIOUSB_FindDevices
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 656
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_FindDevicesByGroup
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 656
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 658
AIOUSB_GetAllDevices
AIODeviceTable.c, 472
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 484
AIOUSB_GetCalMode
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 607
AIOUSB_GetCommTimeout
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_GetConfigBlock
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_GetDevice
AIOUSB_Core.c, 611

AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_GetDeviceByProductID
AIOUSB_Properties.c, 651
AIOUSB_Properties.h, 655
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 653
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_GetDeviceProperties
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 655
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_GetDeviceSerialNumber
AIOUSB_Core.h, 616
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 655
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.h, 620
ndkbuild/jni/AIOUSB_Properties.c, 653
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_GetEndChannel
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_GetGainCode
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 593
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 581
ndkbuild/jni/AIOUSB_ADC.h, 605
AIOUSB_GetMiscClock
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_GetOversample
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_GetRegister
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 598
libaiousb.h, 742
ndkbuild/jni/AIOUSB_ADC.c, 581
ndkbuild/jni/AIOUSB_ADC.h, 609
AIOUSB_GetResultCodeAsString
AIOUSB_Core.h, 617
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 655
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.h, 621
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_GetScan
AIOUSB_ADC.c, 559
AIOUSB_ADC.h, 588
libaiousb.h, 734
ndkbuild/jni/AIOUSB_ADC.c, 572
ndkbuild/jni/AIOUSB_ADC.h, 601
AIOUSB_GetStartChannel
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_GetStreamingBlockSize
AIOUSB_Core.c, 611
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 622
AIOUSB_GetTriggerMode
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_GetVersion
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_GetVersionDate
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_INFO
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_INIT_PATTERN
libaiousb.h, 748
AIOUSB_Init
AIODeviceTable.c, 473
AIODeviceTable.h, 479
libaiousb.h, 730
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 483
AIOUSB_InitConfigBlock
AIOUSB_Core.c, 611
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 623
AIOUSB_InitTest
AIODeviceTable.c, 470
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_IsDifferentialMode
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 593
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 606
AIOUSB_IsDiscardFirstSample
AIOUSB_ADC.c, 566
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 580
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_IsInit
AIODeviceTable.c, 470
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIODeviceTable.h, 483
AIOUSB_LOG
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_ListDevices
AIOUSB_Properties.c, 652

AIOUSB_Properties.h, 655
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_Lock
AIOUSB_Core.c, 610
AIOUSB_Core.h, 616
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_Log.c
cont_thread, 646
message_lock, 646
outfile, 646
AIOUSB_Log.h
AIOUSB_DEBUG, 648
AIOUSB_DEVEL, 648
AIOUSB_ERROR, 648
AIOUSB_FATAL, 648
AIOUSB_INFO, 648
AIOUSB_LOG, 648
AIOUSB_WARN, 648
CYAN, 647
cont_thread, 648
GREEN, 647
LOG_LEVEL, 648
MAGENTA, 647
message_lock, 649
outfile, 649
RED, 647
AIOUSB_Properties.c
AIOUSB_FindDevices, 652
AIOUSB_FindDevicesByGroup, 652
AIOUSB_GetDeviceProperties, 652
AIOUSB_GetDeviceSerialNumber, 652
AIOUSB_ListDevices, 652
AIOUSB_ShowDevices, 652
GetDeviceBySerialNumber, 652
GetDeviceSerialNumber, 652
non_usb_supported_device, 651
AIOUSB_Properties.h
AIODisplayType, 655
AIOUSB_FindDevices, 656
AIOUSB_FindDevicesByGroup, 656
AIOUSB_GetDeviceProperties, 655
AIOUSB_GetDeviceSerialNumber, 655
AIOUSB_ListDevices, 655
AIOUSB_ShowDevices, 656
GetDeviceBySerialNumber, 655
GetDeviceSerialNumber, 655
AIOUSB_Reset
AIODeviceTable.c, 473
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 483
AIOUSB_ResetChip
AIOUSB_Core.c, 610
AIOUSB_Core.h, 616
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_SetAllGainCodeAndDiffMode
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 593
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 581
ndkbuild/jni/AIOUSB_ADC.h, 605
AIOUSB_SetCalMode
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 607
AIOUSB_SetCommTimeout
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_SetDifferentialMode
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 594
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 607
AIOUSB_SetDiscardFirstSample
AIOUSB_ADC.c, 566
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 580
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_SetGainCode
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 593
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 582
ndkbuild/jni/AIOUSB_ADC.h, 605
AIOUSB_SetInit
AIODeviceTable.c, 470
ndkbuild/jni/AIODeviceTable.c, 474
AIOUSB_SetMiscClock
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_SetOversample
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_SetRangeSingle
AIOUSB_ADC.c, 567
ndkbuild/jni/AIOUSB_ADC.c, 581
AIOUSB_SetRegister
AIOUSB_ADC.c, 567
AIOUSB_ADC.h, 598
libaiousb.h, 742
ndkbuild/jni/AIOUSB_ADC.c, 581
ndkbuild/jni/AIOUSB_ADC.h, 609
AIOUSB_SetScanRange
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_SetStreamingBlockSize
AIOUSB_ADC.h, 596
AIOUSB_Core.c, 611
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.h, 608
ndkbuild/jni/AIOUSB_Core.c, 614
AIOUSB_SetTriggerMode
AIOUSB_ADC.c, 568
AIOUSB_ADC.h, 596
libaiousb.h, 740
ndkbuild/jni/AIOUSB_ADC.c, 583
ndkbuild/jni/AIOUSB_ADC.h, 608
AIOUSB_ShowDevices
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 656
libaiousb.h, 744

ndkbuild/jni/AIOUSB_Properties.c, 654
ndkbuild/jni/AIOUSB_Properties.h, 657
AIOUSB_USB.h
usb_bulk_xfer, 658
usb_close, 658
usb_control_xfer, 658
usb_free_devices, 658
usb_get_devices, 658
usb_open, 658
AIOUSB_UnLock
AIOUSB_Core.c, 610
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_Validate
AIOUSB_Core.c, 610
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_Validate_Device
AIOUSB_Core.c, 611
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 623
AIOUSB_Validate_Lock
AIOUSB_Core.c, 610
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
AIOUSB_VoltsToCounts
AIOUSB_ADC.c, 560
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_ADC.c, 573
ndkbuild/jni/AIOUSB_Core.h, 621
AIOUSB_WARN
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
AIOUSB_WDG.c
DeleteWDGConfig, 659
doSomething, 659
NewWDGConfig, 659
WDG_GetStatus, 659
WDG_Pet, 661
WDG_SetConfig, 659
AIOUSB_WDG.h
DeleteWDGConfig, 663
NewWDGConfig, 663
WDG_GetStatus, 663
WDG_Pet, 663
WDG_SetConfig, 663
WDGVals, 663
AIOUSB_WorkFn
AIOContinuousBuffer.h, 437
ndkbuild/jni/AIOContinuousBuffer.h, 445
AIOUSBDevice, 164
ADBuf, 168
ADBuf_size, 168
ADCChannels, 166
ADCChannelsPerGroup, 166
ADCMUXChannels, 166
ADCConfigBlock.h, 367
bADCStream, 166
bClearFIFO, 167
bDACAborting, 167
bDACBoardRange, 167
bDACChannelCal, 167
bDACClosing, 167
bDACDIOClock, 166
bDACDIOStream, 166
bDACOpen, 167
bDACSlowWaveStream, 166
bDACStarted, 167
bDACStream, 166
bDIODebounce, 167
bDIOOpen, 167
bDIORead, 167
bDIOSPI, 167
bDIOStream, 166
bDeviceWasHere, 167
bGateSelectable, 166
bGetName, 166
bOpen, 166
bSetCustomClocks, 167
cachedConfigBlock, 167
cachedName, 167
cachedSerialNumber, 167
commTimeout, 166
ConfigBytes, 166
Counters, 166
DACData, 167
DACsUsed, 166
DIOBytes, 166
DIOConfigBits, 166
deviceIndex, 166
discardFirstSample, 166
FastITBakConfig, 168
FastITConfig, 168
FastITConfig_size, 168
FlashSectors, 167
hDACDataMutex, 167
hDACDataSem, 167
ImmADCs, 167
ImmDACs, 166
isInit, 166
LastDIOData, 167
miscClockHz, 166
ndkbuild/jni/ADCConfigBlock.h, 372
PID, 166
PendingDACData, 167
ProductID, 166
RangeShift, 166
RootClock, 166
StreamingBlockSize, 167
testing, 168
Tristates, 166
usb_device, 166
valid, 168
WDGBytes, 167
workerBusy, 167
workerResult, 167
workerStatus, 167
AIOUSBDevice.c
AIOUSBDeviceCopyADCConfigBlock, 666
AIOUSBDeviceGetADCConfigBlock, 666
AIOUSBDeviceGetDiscardFirstSample, 666
AIOUSBDeviceGetStreamingBlockSize, 666
AIOUSBDeviceGetTesting, 666
AIOUSBDeviceGetTimeout, 666
AIOUSBDeviceGetUSBHandle, 666
AIOUSBDeviceInitializeWithProductID, 666
AIOUSBDeviceSetADCConfigBlock, 666
AIOUSBDeviceSetDiscardFirstSample, 666
AIOUSBDeviceSetTesting, 666
AIOUSBDeviceSetTimeout, 666
AIOUSBDeviceSetUSBHandle, 666
AIOUSBDeviceSize, 666
AIOUSBDeviceToJSON, 666
AIOUSBDeviceWriteADCConfig, 666
DeleteAIOUSBDevice, 666

NewAIOUSBDevice, 666
NewAIOUSBDeviceFromJSON, 666
AIOUSBDevice.h
AIOUSBDeviceCopyADCConfigBlock, 668
AIOUSBDeviceGetADCConfigBlock, 668
AIOUSBDeviceGetDiscardFirstSample, 668
AIOUSBDeviceGetStreamingBlockSize, 668
AIOUSBDeviceGetTesting, 668
AIOUSBDeviceGetTimeout, 668
AIOUSBDeviceGetUSBHandle, 668
AIOUSBDeviceInitializeWithProductID, 668
AIOUSBDeviceSetADCConfigBlock, 668
AIOUSBDeviceSetDiscardFirstSample, 668
AIOUSBDeviceSetTesting, 668
AIOUSBDeviceSetTimeout, 668
AIOUSBDeviceSetUSBHandle, 668
AIOUSBDeviceSize, 668
AIOUSBDeviceToJSON, 668
AIOUSBDeviceWriteADCConfig, 668
DeviceDescriptor, 668
NewAIOUSBDeviceFromJSON, 668
AIOUSBDeviceCopyADCConfigBlock
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceGetADCConfigBlock
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceGetDiscardFirstSample
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceGetStreamingBlockSize
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceGetTesting
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceGetTimeout
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 744
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceGetUSBHandle
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceGetUSBHandleFromDeviceIndex
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceInitializeWithProductID
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceSetADCConfigBlock
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceSetDiscardFirstSample
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 744
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceSetTesting
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceSetTimeout
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 744
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceSetUSBHandle
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceSize
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBDeviceToJSON
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
AIOUSBDeviceWriteADCConfig
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 744
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 670
AIOUSBGetError
AIODeviceTable.c, 472
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 484
AIOWDGConfig, 168
bufsize, 168
L, 168
timeout, 168
wdgbuf, 168
ALERT_LEVEL
TestCaseSetup.h, 851
AO16_AnalogOutputSubsystem, 192
AIOUSB::AO16_AnalogOutputSubsystem, 193
AIOUSB::AO16_OutputRange, 196
AO16_OutputRange, 195
AIOUSB::AO16_OutputRange, 196
API_EXPORTED
libusbi.h, 782
ARRAYSIZE

fxload.c, 761
libusbi.h, 783
windows_common.h, 810
AUR_CBUF_EXIT
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AUR_CBUF_SETUP
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
AUTO_CLEAR_STALL
windows_winusb.h, 826
AUTO_FLUSH
windows_winusb.h, 826
AUTO_SUSPEND
windows_winusb.h, 826
abort_control
windows_usb_api_backend, 340
abort_transfers
windows_usb_api_backend, 340
AbortPipe
winusb_interface, 341
active_config
darwin_cached_device, 214
linux_device_priv, 259
windows_device_priv, 338
active_contexts_list
core.c, 769
libusbi.h, 786
active_contexts_lock
core.c, 769
libusbi.h, 786
active_interface
windows_device_handle_priv, 337
active_signals
AIOChannelMask, 156
ActiveConfiguration
USBDevice, 313
ActiveConfigurationIndex
USBDevice, 313
actual_length
libusb_iso_packet_descriptor, 251
libusb_transfer, 256
usbfs_iso_packet_desc, 328
usbfs_urb, 330
actual_size
AIOArgument, 155
ActualLength
tag_USB_DK_ISO_TRANSFER_RESULT, 271
USB_NODE_CONNECTION_NAME, 307
USB_NODE_CONNECTION_NAME_FIXED, 307
USB_ROOT_HUB_NAME, 311
USB_ROOT_HUB_NAME_FIXED, 311
ad_gain_pairs, 145
gain, 145
name, 145
adRanges
AIOUSB_ADC.c, 569
AIOUSB_Core.h, 618
ndkbuild/jni/AIOUSB_ADC.c, 584
ndkbuild/jni/AIOUSB_Core.h, 623
adc
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 282
adc_get_bulk_data
AIOUSB_ADC.c, 559
mock_aiocontbuf_get_data_arduino.c, 694
ndkbuild/jni/AIOUSB_ADC.c, 572
adcblock_valid_channel_settings
AIOUSB_ADC.c, 562
ndkbuild/jni/AIOUSB_ADC.c, 575
adcblock_valid_size
AIOUSB_ADC.c, 562
ndkbuild/jni/AIOUSB_ADC.c, 575
adcblock_valid_trigger_settings
AIOUSB_ADC.c, 561
ndkbuild/jni/AIOUSB_ADC.c, 575
adcconfig_json
AIOCommandLineOptions, 159
opts, 264
add_devices_fn
mock_usb_xfers.c, 698
AddAllACCESUSBDevices
libaiousb.h, 745
mock_usb_xfers.c, 698
ndkbuild/jni/USBDevice.c, 748
ndkbuild/jni/USBDevice.h, 751
USBDevice.c, 750
USBDevice.h, 753
AddDevice
ndkbuild/jni/USBDevice.c, 748
USBDevice.c, 750
address
darwin_cached_device, 214
aio_callback_args, 152
aiocb, 152
transfer, 152
aio_channel_obj
AIOChannelMask.h, 385
ndkbuild/jni/AIOChannelMask.h, 388
aio_channel_range, 152
end, 153
gain, 153
start, 153
aio_channel_range_error
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 748
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
aio_counts_converter, 153
buf, 153
channel_count, 153
continue_conversion, 153
Convert, 154
ConvertFifo, 154
converted_count, 153
discardFirstSample, 154
gain_ranges, 153
num_channels, 153
num_oversamples, 153
num_scans, 153
os_count, 153
scan_count, 153
sum, 153
unit_size, 153
aio_either_val, 154
number, 154
object, 154
aio_errno
AIOUSB_Core.c, 612
AIOUSB_Core.h, 618
ndkbuild/jni/AIOUSB_Core.c, 615
ndkbuild/jni/AIOUSB_Core.h, 623
aio_list_devices
aiocommon.c, 849
aiocommon.h, 850
aio_override_adcconfig_settings
aiocommon.c, 849
aiocommon.h, 850
aio_override_aiobuf_settings
aiocommon.c, 849
aiocommon.h, 850
aio_ret_value, 154
errmsg, 154

left, 154
right, 154
size, 154
type, 154
aio_supply_default_command_line_settings
aiocommon.c, 849
aiocommon.h, 850
aiobuf_iterator, 156
buf, 156
loc, 156
next, 156
aiobuf_json
AIOCommandLineOptions, 159
opts, 264
aiocb
aio_callback_args, 152
sunos_transfer_priv, 269
aiocommon.c
AIO_OPTIONS, 849
aio_list_devices, 849
aio_override_adcconfig_settings, 849
aio_override_aiobuf_settings, 849
aio_supply_default_command_line_settings, 849
CNTS, 849
DUMP, 849
get_channel_range, 849
JCONF, 849
print_aio_usage, 849
process_aio_cmd_line, 849
REPEAT, 849
aiocommon.h
AIO_OPTIONS, 850
aio_list_devices, 850
aio_override_adcconfig_settings, 850
aio_override_aiobuf_settings, 850
aio_supply_default_command_line_settings, 850
get_channel_range, 850
print_aio_usage, 850
process_aio_cmd_line, 850
aiocontbuf_get_bulk_data
mock_aiocontbuf_get_data.c, 693
mock_aiocontbuf_get_data_arduino.c, 694
aioeither_value_double
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_double_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_int
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_int32_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_longdouble_t
AIOEither.h, 487, 488
ndkbuild/jni/AIOEither.h, 489
aioeither_value_obj
AIOEither.h, 487, 488
ndkbuild/jni/AIOEither.h, 489
aioeither_value_string
AIOEither.h, 487, 488
ndkbuild/jni/AIOEither.h, 489
aioeither_value_uint16_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_uint32_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_uint8_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_value_unsigned
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioeither_vlaue_int16_t
AIOEither.h, 487
ndkbuild/jni/AIOEither.h, 489
aioerror, 162
error_message, 162
retval, 162
AIOUSB, 7
aiousb_htons
AIOUSB_DIO.c, 639
ndkbuild/jni/AIOUSB_DIO.c, 640
aiousb_libusb_args, 164
dev, 164
deviceDesc, 164
handle, 164
aiousbInit
AIODeviceTable.c, 473
AIOUSB_Core.h, 618
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIOUSB_Core.h, 623
aiousboption, 168
alloc_streams
usbi_os_backend, 332
alternate
usb_raw_command, 310
alternate_index
usb_raw_command, 310
alternate_info
usb_raw_command, 310
altset
USBDevice, 314
altsetting
libusb_interface, 248
sunos_device_handle_priv, 268
usbfs_setinterface, 329
AnalogIORange, 188
AIOUSB::AnalogIORange, 188
AnalogInputSubsystem, 168
AIOUSB::AI16_DataPoint, 148
AIOUSB::AI16_DataSet, 151
AIOUSB::AI16_InputRange, 152
AIOUSB::AnalogInputSubsystem, 172
AIOUSB::USBDeviceBase, 318
analogInputSubsystem
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AnalogOutputSubsystem, 190
AIOUSB::AnalogOutputSubsystem, 191
analogOutputSubsystem
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_DA12_8A_Family, 291
AIOUSB::USB_DA12_8E_Family, 294
analogOutputSubsytem
AIOUSB::USB_AIO16_Family, 283
android/config.h
DEFAULT_VISIBILITY, 754
ENABLE_LOGGING, 754
HAVE_ASM_TYPES_H, 755
HAVE_DLFCN_H, 754
HAVE_GETTIMEOFDAY, 754
HAVE_INTTYPES_H, 754
HAVE_MEMORY_H, 754
HAVE_POLL_H, 754
HAVE_SIGNAL_H, 754
HAVE_SYS_SOCKET_H, 755
HAVE_SYS_STAT_H, 754
HAVE_SYS_TIME_H, 754
HAVE_SYS_TYPES_H, 754
HAVE_UNISTD_H, 754
OS_LINUX, 754

POLL_NFDS_TYPE, 754
THREADS_POSIX, 754
api_handle
interface_handle_t, 235
apib
windows_device_priv, 338
Asynchronous device I/O
LIBUSB_TRANSFER_ADD_ZERO_PACKET, 126
LIBUSB_TRANSFER_CANCELLED, 126
LIBUSB_TRANSFER_COMPLETED, 126
LIBUSB_TRANSFER_ERROR, 126
LIBUSB_TRANSFER_FREE_BUFFER, 126
LIBUSB_TRANSFER_FREE_TRANSFER, 126
LIBUSB_TRANSFER_NO_DEVICE, 126
LIBUSB_TRANSFER_OVERFLOW, 126
LIBUSB_TRANSFER_SHORT_NOT_OK, 126
LIBUSB_TRANSFER_STALL, 126
LIBUSB_TRANSFER_TIMED_OUT, 126
Asynchronous device I/O, 121
libusb_alloc_streams, 126
libusb_alloc_transfer, 127
libusb_cancel_transfer, 128
libusb_dev_mem_alloc, 127
libusb_dev_mem_free, 127
libusb_free_streams, 127
libusb_free_transfer, 128
libusb_submit_transfer, 128
libusb_transfer_flags, 126
libusb_transfer_get_stream_id, 130
libusb_transfer_set_stream_id, 128
libusb_transfer_status, 126
void, 130
attach_kernel_driver
usbi_os_backend, 332
attached
libusb_device, 242
auto_detach_kernel_driver
libusb_device_handle, 244
autoCalFeature
AIOUSB::AnalogInputSubsystem, 187
autoConfig
AIOUSB::AnalogInputSubsystem, 187
autoclaim_count
windows_device_handle_priv, 337

B
xusb.c, 766
B_USB_RAW_COMMAND_BULK_TRANSFER
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_CLEAR_FEATURE
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_CONTROL_TRANSFER
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_ACTIVE_ALT_INTERFACE_INDEX
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_ALT_INTERFACE_COUNT
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_CONFIGURATION_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_DEVICE_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR_ETC
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR_ETC
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR_ETC
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_STATUS
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_STRING_DESCRIPTOR
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_GET_VERSION
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_INTERRUPT_TRANSFER
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_ISOCHRONOUS_TRANSFER
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_SET_ALT_INTERFACE
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_SET_CONFIGURATION
haiku_usb_raw.h, 789
B_USB_RAW_COMMAND_SET_FEATURE
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_ABORTED
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_CRC_ERROR
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_FAILED
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_INVALID_CONFIGURATION
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_INVALID_ENDPOINT
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_INVALID_INTERFACE
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_INVALID_STRING
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_NO_MEMORY
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_STALLED
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_SUCCESS
haiku_usb_raw.h, 789
B_USB_RAW_STATUS_TIMEOUT
haiku_usb_raw.h, 789
BASIC
AIOUSB_Properties.h, 655
ndkbuild/jni/AIOUSB_Properties.h, 656, 657
BEGIN
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
BITS_PER_BYTE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
BACKTRACE_DEBUG
AIOUSB_Core.c, 610
ndkbuild/jni/AIOUSB_Core.c, 613
bADCStream
AIOUSBDevice, 166
bAlternateSetting
libusb_interface_descriptor, 249
USB_INTERFACE_DESCRIPTOR, 305
bCBWCBLength
command_block_wrapper, 198

bCBWLUN
command_block_wrapper, 198
bCSWStatus
command_status_wrapper, 199
bClassDescriptorType
libusb_hid_descriptor, 246
bClearFIFO
AIOUSBDevice, 167
bConfigurationValue
libusb_config_descriptor, 237
USB_CONFIGURATION_DESCRIPTOR, 286
bCountryCode
libusb_hid_descriptor, 246
bDACAborting
AIOUSBDevice, 167
bDACBoardRange
AIOUSBDevice, 167
bDACChannelCal
AIOUSBDevice, 167
bDACClosing
AIOUSBDevice, 167
bDACDIOClock
AIOUSBDevice, 166
bDACDIOStream
AIOUSBDevice, 166
bDACOpen
AIOUSBDevice, 167
bDACSlowWaveStream
AIOUSBDevice, 166
bDACStarted
AIOUSBDevice, 167
bDACStream
AIOUSBDevice, 166
bDIODebounce
AIOUSBDevice, 167
bDIOOpen
AIOUSBDevice, 167
bDIORead
AIOUSBDevice, 167
bDIOSPI
AIOUSBDevice, 167
bDIOStream
AIOUSBDevice, 166
bDescriptorLength
USB_HUB_DESCRIPTOR, 304
bDescriptorType
libusb_bos_descriptor, 235
libusb_bos_dev_capability_descriptor, 236
libusb_config_descriptor, 237
libusb_container_id_descriptor, 238
libusb_device_descriptor, 243
libusb_endpoint_descriptor, 245
libusb_hid_descriptor, 246
libusb_interface_descriptor, 249
libusb_ss_endpoint_companion_descriptor, 252
libusb_ss_usb_device_capability_descriptor, 253
libusb_usb_2_0_extension_descriptor, 257
UKW_DEVICE_DESCRIPTOR, 277
USB_CONFIGURATION_DESCRIPTOR, 286
usb_descriptor_header, 294
USB_ENDPOINT_DESCRIPTOR, 302
USB_HUB_DESCRIPTOR, 304
USB_INTERFACE_DESCRIPTOR, 305
bDevCapabilityType
libusb_bos_dev_capability_descriptor, 236
libusb_container_id_descriptor, 238
libusb_ss_usb_device_capability_descriptor, 253
libusb_usb_2_0_extension_descriptor, 257
bDeviceClass
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
bDeviceProtocol
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
bDeviceSubClass
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
bDeviceWasHere
AIOUSBDevice, 167
bEndpointAddress
libusb_endpoint_descriptor, 245
USB_ENDPOINT_DESCRIPTOR, 302
bFunctionalitySupport
libusb_ss_usb_device_capability_descriptor, 253
bGateSelectable
AIOUSBDevice, 166
bGetName
AIOUSBDevice, 166
bHubControlCurrent
USB_HUB_DESCRIPTOR, 304
BIT_LENGTH
AIOChannelMask.h, 385
ndkbuild/jni/AIOChannelMask.h, 388
BITS_PER_BYTE
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 876
bInterfaceClass
libusb_interface_descriptor, 250
USB_INTERFACE_DESCRIPTOR, 305
bInterfaceNumber
libusb_interface_descriptor, 249
USB_INTERFACE_DESCRIPTOR, 305
bInterfaceProtocol
libusb_interface_descriptor, 250
USB_INTERFACE_DESCRIPTOR, 305
bInterfaceSubClass
libusb_interface_descriptor, 250
USB_INTERFACE_DESCRIPTOR, 305
bInterval
libusb_endpoint_descriptor, 245
USB_ENDPOINT_DESCRIPTOR, 302
bLength
libusb_bos_descriptor, 235
libusb_bos_dev_capability_descriptor, 236
libusb_config_descriptor, 237
libusb_container_id_descriptor, 238
libusb_device_descriptor, 243
libusb_endpoint_descriptor, 245
libusb_hid_descriptor, 246
libusb_interface_descriptor, 249
libusb_ss_endpoint_companion_descriptor, 252
libusb_ss_usb_device_capability_descriptor, 253
libusb_usb_2_0_extension_descriptor, 257
UKW_DEVICE_DESCRIPTOR, 277
USB_CONFIGURATION_DESCRIPTOR, 286
usb_descriptor_header, 294
USB_ENDPOINT_DESCRIPTOR, 302
USB_INTERFACE_DESCRIPTOR, 305
BLooper, 196
bMaxBurst
libusb_ss_endpoint_companion_descriptor, 252
bMaxPacketSize0
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
bNumConfigurations
libusb_device_descriptor, 244
UKW_DEVICE_DESCRIPTOR, 278
bNumDescriptors
libusb_hid_descriptor, 246
bNumDeviceCaps
libusb_bos_descriptor, 235

bNumEndpoints
libusb_interface_descriptor, 250
USB_INTERFACE_DESCRIPTOR, 305
bNumInterfaces
libusb_config_descriptor, 237
USB_CONFIGURATION_DESCRIPTOR, 286
bNumberOfPorts
USB_HUB_DESCRIPTOR, 304
BOMS_GET_MAX_LUN
xusb.c, 767
BOMS_RESET
xusb.c, 767
bOpen
AIOUSBDevice, 166
bPowerOnToPowerGood
USB_HUB_DESCRIPTOR, 304
bRefresh
libusb_endpoint_descriptor, 245
bRemoveAndPowerMask
USB_HUB_DESCRIPTOR, 304
bRequest
libusb_control_setup, 241
UKW_CONTROL_HEADER, 277
USB_CONFIGURATION_DESCRIPTOR_SHORT, 286
USB_DESCRIPTOR_REQUEST, 295
usbfs_ctrltransfer, 327
bReserved
libusb_container_id_descriptor, 239
bSetCustomClocks
AIOUSBDevice, 167
bSynchAddress
libusb_endpoint_descriptor, 245
bU1DevExitLat
libusb_ss_usb_device_capability_descriptor, 253
bU2DevExitLat
libusb_ss_usb_device_capability_descriptor, 253
base_size
AIOContinuousBuf, 160
BaseSizeRange
AIOContinuousBuffer.c, 425
ndkbuild/jni/AIOContinuousBuffer.c, 434
bcdDevice
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
bcdHID
libusb_hid_descriptor, 246
bcdUSB
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
be_to_int32
xusb.c, 766
bitsToBytes
AIOUSB::DigitalIOSubsystem, 220
block_size
AIOCommandLineOptions, 159
AIOContinuousBuf, 160
opts, 264
TestCaseSetup, 275
bmAttributes
libusb_config_descriptor, 237
libusb_endpoint_descriptor, 245
libusb_ss_endpoint_companion_descriptor, 252
libusb_ss_usb_device_capability_descriptor, 253
libusb_usb_2_0_extension_descriptor, 257
USB_CONFIGURATION_DESCRIPTOR, 286
USB_ENDPOINT_DESCRIPTOR, 302
bmCBWFlags
command_block_wrapper, 198
bmRequest
USB_CONFIGURATION_DESCRIPTOR_SHORT, 286
USB_DESCRIPTOR_REQUEST, 295
bmRequestType
libusb_control_setup, 241
UKW_CONTROL_HEADER, 277
usbfs_ctrltransfer, 327
bool
libusb_testlib.h, 846
windows_common.h, 809
xusb.c, 766
BoolArray, 196
AIOUSB::BoolArray, 196
buf
aio_counts_converter, 153
aiobuf_iterator, 156
BufSize
BulkAcquireWorkerParams, 197
Buffer
tag_USB_DK_TRANSFER_REQUEST, 271
buffer
AIOContinuousBuf, 160
DIOBuf, 225
libusb_transfer, 256
usbfs_urb, 330
buffer_length
usbfs_urb, 330
buffer_size
AIOCommandLineOptions, 159
config_options, 199
opts, 264
BufferLength
tag_USB_DK_TRANSFER_REQUEST, 271
bufsize
AIOWDGConfig, 168
BulkTransferType
windows_usbdk.h, 816
bulk_acquire_sample.c
main, 853
print_usage, 853
process_cmd_line, 853
BulkAcquireWorkerParams, 197
BufSize, 197
DeviceIndex, 197
pBuf, 197
burst_test.c
find_ai_board, 854
bus_number
libusb_device, 242
bytes_processed
AIOContinuousBuf, 160
bytesToBits
AIOUSB::DigitalIOSubsystem, 220
BytesTransferred
tag_USB_DK_GEN_TRANSFER_RESULT, 270

CAL_TABLE_WORDS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
CANCELLED
linux_usbfs.c, 792
CHANNEL_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
CLEAR_FIFO_METHOD_AUTO
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
CLEAR_FIFO_METHOD_IMMEDIATE
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
CLEAR_FIFO_METHOD_NOW

AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
CLEAR_FIFO_METHOD_WAIT
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
COMPLETED_EARLY
linux_usbfs.c, 792
COUNT_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
COUNTER_NUM_MODES
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
COUNTERS_PER_BLOCK
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
CYPRESS_DESC_PARAMS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
CYPRESS_GET_DESC
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
CYPRESS_MAX_DESC_SIZE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
CAL_CHANNEL
TestCaseSetup, 275
CAL_MODE_GROUND
AIOUSB::AnalogInputSubsystem, 185
CAL_MODE_NORMAL
AIOUSB::AnalogInputSubsystem, 185
CAL_TABLE_WORDS
AIOUSB::AnalogInputSubsystem, 186
CALL_CHECK
xusb.c, 766
CBWCB
command_block_wrapper, 199
CHECK_INIT_POLLING
poll_windows.c, 797
CHECK_RESULT
read_and_write.c, 871
TestCaseSetup.h, 852
cJSON, 197
child, 198
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
next, 198
prev, 198
string, 198
type, 198
valuedouble, 198
valueint, 198
valuestring, 198
cJSON.c
cJSON_AddItemReferenceToArray, 671
cJSON_AddItemReferenceToObject, 671
cJSON_AddItemToArray, 671
cJSON_AddItemToObject, 671
cJSON_AsInteger, 671
cJSON_CreateArray, 671
cJSON_CreateBool, 671
cJSON_CreateDoubleArray, 671
cJSON_CreateFalse, 671
cJSON_CreateFloatArray, 671
cJSON_CreateIntArray, 671
cJSON_CreateNull, 671
cJSON_CreateNumber, 671
cJSON_CreateObject, 671
cJSON_CreateString, 671
cJSON_CreateStringArray, 672
cJSON_CreateTrue, 671
cJSON_Delete, 671
cJSON_DeleteItemFromArray, 671
cJSON_DeleteItemFromObject, 671
cJSON_DetachItemFromArray, 671
cJSON_DetachItemFromObject, 671
cJSON_Duplicate, 672
cJSON_GetArrayItem, 671
cJSON_GetArraySize, 671
cJSON_GetErrorPtr, 671
cJSON_GetObjectItem, 671
cJSON_InitHooks, 671
cJSON_Minify, 672
cJSON_Parse, 671
cJSON_ParseWithOpts, 671
cJSON_Print, 671
cJSON_PrintUnformatted, 671
cJSON_ReplaceItemInArray, 671
cJSON_ReplaceItemInObject, 671
cJSON.h
cJSON, 675
cJSON_AddBoolToObject, 675
cJSON_AddFalseToObject, 675
cJSON_AddItemReferenceToArray, 676
cJSON_AddItemReferenceToObject, 676
cJSON_AddItemToArray, 676
cJSON_AddItemToObject, 676
cJSON_AddNullToObject, 675
cJSON_AddNumberToObject, 675
cJSON_AddStringToObject, 675
cJSON_AddTrueToObject, 675
cJSON_Array, 675
cJSON_AsInteger, 675
cJSON_CreateArray, 676
cJSON_CreateBool, 676
cJSON_CreateDoubleArray, 676
cJSON_CreateFalse, 676
cJSON_CreateFloatArray, 676
cJSON_CreateIntArray, 676
cJSON_CreateNull, 676
cJSON_CreateNumber, 676
cJSON_CreateObject, 676
cJSON_CreateString, 676
cJSON_CreateStringArray, 676
cJSON_CreateTrue, 676
cJSON_Delete, 675
cJSON_DeleteItemFromArray, 676
cJSON_DeleteItemFromObject, 676
cJSON_DetachItemFromArray, 676
cJSON_DetachItemFromObject, 676
cJSON_Duplicate, 676
cJSON_False, 675
cJSON_GetArrayItem, 675
cJSON_GetArraySize, 675
cJSON_GetErrorPtr, 675
cJSON_GetObjectItem, 675
cJSON_Hooks, 675
cJSON_InitHooks, 675
cJSON_IsReference, 675
cJSON_Minify, 676
cJSON_NULL, 675
cJSON_Number, 675
cJSON_Object, 675
cJSON_Parse, 675
cJSON_ParseWithOpts, 676
cJSON_Print, 675
cJSON_PrintUnformatted, 675
cJSON_ReplaceItemInArray, 676
cJSON_ReplaceItemInObject, 676
cJSON_SetIntValue, 675
cJSON_String, 675
cJSON_True, 675
cJSON_AddBoolToObject
cJSON.h, 675

ndkbuild/jni/cJSON.h, 678
cJSON_AddFalseToObject
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_AddItemReferenceToArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_AddItemReferenceToObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_AddItemToArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_AddItemToObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_AddNullToObject
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_AddNumberToObject
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_AddStringToObject
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_AddTrueToObject
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_Array
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_AsInteger
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_CreateArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateBool
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_CreateDoubleArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateFalse
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_CreateFloatArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateIntArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateNull
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_CreateNumber
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_CreateObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateString
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateStringArray
cJSON.c, 672
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_CreateTrue
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_Delete
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_DeleteItemFromArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_DeleteItemFromObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_DetachItemFromArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_DetachItemFromObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_Duplicate
cJSON.c, 672
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_False
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_GetArrayItem
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_GetArraySize
cJSON.c, 671

cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_GetErrorPtr
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_GetObjectItem
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_Hooks, 198
cJSON.h, 675
free_fn, 198
malloc_fn, 198
ndkbuild/jni/cJSON.h, 678
cJSON_InitHooks
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_IsReference
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_Minify
cJSON.c, 672
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_NULL
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_Number
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_Object
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_Parse
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_ParseWithOpts
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_Print
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_PrintUnformatted
cJSON.c, 671
cJSON.h, 675
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 678
cJSON_ReplaceItemInArray
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_ReplaceItemInObject
cJSON.c, 671
cJSON.h, 676
ndkbuild/jni/cJSON.c, 673
ndkbuild/jni/cJSON.h, 679
cJSON_SetIntValue
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_String
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
cJSON_True
cJSON.h, 675
ndkbuild/jni/cJSON.h, 678
CNTS
AIOCommandLine.h, 404
aiocommon.c, 849
ndkbuild/jni/AIOCommandLine.h, 407
CONFIG_DESC_LENGTH
descriptor.c, 771
CONFIGRET
windows_winusb.h, 827
COUNTS
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
CR_SUCCESS
windows_winusb.h, 825
CREATE_ENUM
AIOTypes.h, 517
ndkbuild/jni/AIOTypes.h, 539
CStringArray, 207
CStringArray.h, 681
ndkbuild/jni/CStringArray.h, 682
CStringArray.c
CStringArrayToString, 680
CStringArrayToStringWithDelimeter, 680
CopyCStringArray, 680
DeleteCStringArray, 680
NewCStringArray, 679
NewCStringArrayFromCArgs, 679
NewCStringArrayWithStrings, 679
CStringArray.h
CStringArray, 681
CStringArrayToString, 681
CStringArrayToStringWithDelimeter, 681
CopyCStringArray, 681
DeleteCStringArray, 681
NewCStringArray, 681
NewCStringArrayFromCArgs, 681
NewCStringArrayWithStrings, 681
STRING_ARRAY, 681
CStringArrayToString
CStringArray.c, 680
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
CStringArrayToStringWithDelimeter
CStringArray.c, 680
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
CTL_CODE
windows_winusb.h, 825
CTR_8254Load
AIOUSB_CTR.c, 624
AIOUSB_CTR.h, 627
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254Mode
AIOUSB_CTR.c, 624
AIOUSB_CTR.h, 627
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 628
CTR_8254ModeLoad
AIOUSB_CTR.c, 624
AIOUSB_CTR.h, 627
libaiousb.h, 721

ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254Read
AIOUSB_CTR.c, 624
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254ReadAll
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254ReadLatched
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 627
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254ReadModeLoad
AIOUSB_CTR.c, 624
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254ReadStatus
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_8254SelectGate
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 627
ndkbuild/jni/AIOUSB_CTR.h, 629
CTR_CalculateCountersForClock
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 627
libaiousb.h, 720
ndkbuild/jni/AIOUSB_CTR.c, 626
ndkbuild/jni/AIOUSB_CTR.h, 628
CTR_StartOutputFreq
AIOUSB_CTR.c, 625
AIOUSB_CTR.h, 628
libaiousb.h, 721
ndkbuild/jni/AIOUSB_CTR.c, 627
ndkbuild/jni/AIOUSB_CTR.h, 629
CTRL_IN
dpfp.c, 756
dpfp_threaded.c, 757
CTRL_OUT
dpfp.c, 756
dpfp_threaded.c, 757
CYAN
AIOUSB_Log.h, 647
ndkbuild/jni/AIOUSB_Log.h, 649
cachedConfigBlock
AIOUSBDevice, 167
cachedName
AIOUSBDevice, 167
cachedSerialNumber
AIOUSBDevice, 167
cal_channel
opts, 264
calMode
AIOUSB::AI16_DataSet, 151
AIOUSB::AnalogInputSubsystem, 187
calibrate
AIOUSB::AnalogInputSubsystem, 180, 181
calibration
AIOCommandLineOptions, 159
configuration, 200
opts, 264
calibration_enabled
opts, 264
TestCaseSetup, 275
calibration_file
configuration, 200
callback
AIOContinuousBuf, 160
libusb_transfer, 256
can_enumerate
darwin_cached_device, 214
cancel_fn
winfd, 340
cancel_transfer
poll_windows.h, 800
usbi_os_backend, 332
CancelTransfer
USBDeviceHandle, 319
CapabilityFlags
USB_HUB_CAPABILITIES_EX, 304
capacity
discovered_devs, 229
caps
linux_device_handle_priv, 258
usbi_os_backend, 332
capture_data
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 855
dirktest.c, 857
test.c, 860
cb
libusb_hotplug_callback, 247
cdesc
device_priv, 216
cfSource
darwin_device_handle_priv, 214
darwin_device_handle_priv::darwin_interface, 215
cfgvalue
sunos_device_priv, 269
channel
AIOCmd, 157
AIOUSB::AI16_DataPoint, 148
AIOUSB::OutputVoltagePoint, 265
channel_count
aio_counts_converter, 153
channel_range, 197
end_channel, 197
gaincode, 197
start_channel, 197
channelsPerGroup
AIOUSB::AnalogInputSubsystem, 187
CheckInterfacesFree
USBDevice, 313
child
cJSON, 198
claim_interface
usbi_os_backend, 332
windows_usb_api_backend, 339
ClaimInterface
USBDevice, 313
USBDeviceHandle, 319
claimed_interfaces
libusb_device_handle, 244
clear_halt
usbi_os_backend, 332
windows_usb_api_backend, 340
clear_transfer_priv
usbi_os_backend, 332
ClearAIODeviceTable

AIODeviceTable.c, 472
AIODeviceTable.h, 480
libaiousb.h, 731
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 483
ClearDevices
AIODeviceTable.c, 472
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 481
clearFIFO
AIOUSB::AnalogInputSubsystem, 183
AIOUSB::DIOStreamSubsystem, 228
AIOUSB::USBDeviceBase, 315
clock_gettime
usbi_os_backend, 332
clock_rate
ADCConfigBlock, 146
AIOCommandLineOptions, 158
config_options, 199
opts, 264
clock_scale
opts, 264
clock_speed
opts, 264
TestCaseSetup, 275
clockHz
AIOUSB::DIOStreamSubsystem, 228
close
AIOUSB::DIOStreamSubsystem, 227
AIOUSB::USBDeviceManager, 324
usbi_os_backend, 332
windows_usb_api_backend, 339
CloseAllDevices
AIODeviceTable.c, 472
AIODeviceTable.h, 480
libaiousb.h, 731
mock_aiocontbuf_get_data_arduino.c, 694
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 483
commTimeout
AIOUSBDevice, 166
command_block_wrapper, 198
bCBWCBLength, 198
bCBWLUN, 198
bmCBWFlags, 198
CBWCB, 199
dCBWDataTransferLength, 198
dCBWSignature, 198
dCBWTag, 198
command_status_wrapper, 199
bCSWStatus, 199
dCSWDataResidue, 199
dCSWSignature, 199
dCSWTag, 199
completed_list
usbi_transfer, 333
completed_transfers
libusb_context, 240
conf
USBDevice, 313
config
AIOArgument, 155
hid_device_priv, 231
usb_raw_command, 310
config_descriptor
windows_device_priv, 338
config_index
sunos_device_handle_priv, 268
usb_raw_command, 310
config_options, 199
buffer_size, 199
clock_rate, 199
framePoints, 199
number_channels, 199
targetSerialNumber, 199
write_clock_rate, 199
configBlock
TestCaseSetup, 275
configBlockSize
AIOUSB::AnalogInputSubsystem, 187
ConfigBytes
AIOUSBDevice, 166
configuration, 199
calibration, 200
calibration_file, 200
configure, 200
debug, 200
device_index, 200
discard_first_sample, 200
file_handle, 200
file_name, 200
number_scans, 200
output_file, 200
run, 200
scan_type, 200
timeout, 200
type, 200
ConfigurationDescriptor
USBDevice, 313
ConfigurationType
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413
configure
AIOUSB::DigitalIOSubsystem, 221, 222
configuration, 200
configure_endpoints
windows_usb_api_backend, 339
ConfigureAndBulkAcquire
AIOUSB_ADC.c, 567
ndkbuild/jni/AIOUSB_ADC.c, 581
ConnectionIndex
USB_CONFIGURATION_DESCRIPTOR_SHORT, 286
USB_DESCRIPTOR_REQUEST, 295
USB_NODE_CONNECTION_INFORMATION_EX, 306
USB_NODE_CONNECTION_NAME, 307
USB_NODE_CONNECTION_NAME_FIXED, 307
ConnectionStatus
USB_NODE_CONNECTION_INFORMATION_EX, 306
const
getopt.c, 762
getopt1.c, 763
cont_thread
AIOUSB_Log.c, 646
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.c, 647
ndkbuild/jni/AIOUSB_Log.h, 650
container_of
libusbi.h, 783
ContainerID
libusb_container_id_descriptor, 239
continue_conversion
aio_counts_converter, 153
continue_running
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
continuous_mode.c
fnd, 854
continuous_mode_callback.c
capture_data, 855

fnd, 855
fp, 855
get_channel_range, 855
main, 855
process_cmd_line, 855
continuous_mode_from_json_config.c
capture_data, 855
fnd, 855
fp, 856
get_channel_range, 855
main, 856
process_cmd_line, 855
control
usb_raw_command, 311
ControlTransferType
windows_usbdk.h, 816
ControlTransfer
winusb_interface, 341
Convert
aio_counts_converter, 154
AIOCountsConverter.c, 452
ndkbuild/jni/AIOCountsConverter.c, 453
ConvertCountsToVoltsFunction
AIOContinuousBuffer.c, 417
ndkbuild/jni/AIOContinuousBuffer.c, 428
ConvertFifo
aio_counts_converter, 154
converted_count
aio_counts_converter, 153
copy_transfer_data
windows_usb_api_backend, 340
CopyCStringArray
CStringArray.c, 680
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
CopyUSBDevice
libaiousb.h, 745
ndkbuild/jni/USBDevice.c, 748
ndkbuild/jni/USBDevice.h, 751
USBDevice.c, 750
USBDevice.h, 753
core.c
active_contexts_list, 769
active_contexts_lock, 769
discovered_devs_append, 769
usbi_alloc_device, 769
usbi_clear_event, 769
usbi_connect_device, 769
usbi_default_context, 769
usbi_disconnect_device, 769
usbi_get_device_by_session_id, 769
usbi_log, 769
usbi_log_v, 769
usbi_sanitize_device, 769
usbi_signal_event, 769
count
ram_poke_context, 267
CountConfigurations
USBDevice, 313
Counter, 200
AIOUSB::Counter, 201
AIOUSB::CounterSubsystem, 207
counter_control
AIOContinuousBuf, 161
counterIndex
AIOUSB::Counter, 203
CounterList, 204
CounterSubsystem, 204
AIOUSB::Counter, 203
AIOUSB::CounterSubsystem, 205
counterSubsystem
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_CTR_15_Family, 289
AIOUSB::USB_DIO_32_Family, 300
Counters
AIODeviceInfo, 161
AIOUSBDevice, 166
DeviceProperties, 217
counters
AIOUSB::CounterSubsystem, 207
counts
AIOCommandLineOptions, 159
AIOUSB::AI16_DataPoint, 148
opts, 264
TestCaseSetup, 275
countsToVolts
AIOUSB::AnalogInputSubsystem, 184
AIOUSB::AnalogIORange, 189
AIOUSB::AO16_AnalogOutputSubsystem, 194
AIOUSB::DA12_AnalogOutputSubsystem, 211
countsbuf
AIOContinuousBuf, 160
CreateSmartBuffer
AIOUSB_ADC.c, 564
AIOUSB_ADC.h, 596
libaiousb.h, 741
ndkbuild/jni/AIOUSB_ADC.c, 579
ndkbuild/jni/AIOUSB_ADC.h, 608
cstuff, 11
ctr
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_CTR_15_Family, 289
AIOUSB::USB_DIO_32_Family, 299
ctx
libusb_device, 242
libusb_hotplug_callback, 247
node_args, 261
CustomEEPROMRead
AIOUSB_CustomEEPROM.c, 630
AIOUSB_CustomEEPROM.h, 631
libaiousb.h, 743
ndkbuild/jni/AIOUSB_CustomEEPROM.c, 630
ndkbuild/jni/AIOUSB_CustomEEPROM.h, 632
customEEPROMRead
AIOUSB::USBDeviceBase, 317
CustomEEPROMWrite
AIOUSB_CustomEEPROM.c, 630
AIOUSB_CustomEEPROM.h, 631
libaiousb.h, 743
ndkbuild/jni/AIOUSB_CustomEEPROM.c, 630
ndkbuild/jni/AIOUSB_CustomEEPROM.h, 631
customEEPROMWrite
AIOUSB::USBDeviceBase, 317

DAC_RANGE_0_10V
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
DAC_RANGE_0_5V
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
DAC_RANGE_10V
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
DAC_RANGE_5V
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
DAC_RESET
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
DACRange_begin
AIOTypes.h, 526

ndkbuild/jni/AIOTypes.h, 548
DACRange_end
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
DEBUG_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
DA12_AnalogOutputSubsystem, 207
AIOUSB::DA12_AnalogOutputSubsystem, 208
AIOUSB::DA12_OutputRange, 213
DA12_OutputRange, 212
AIOUSB::DA12_OutputRange, 213
DACChannels
DeviceProperties, 218
DACData
AIOUSBDevice, 167
DACDirect
AIOUSB_DAC.c, 632
AIOUSB_DAC.h, 635
libaiousb.h, 742
ndkbuild/jni/AIOUSB_DAC.c, 634
ndkbuild/jni/AIOUSB_DAC.h, 637
DACMultiDirect
AIOUSB_DAC.c, 632
AIOUSB_DAC.h, 635
libaiousb.h, 742
ndkbuild/jni/AIOUSB_DAC.c, 634
ndkbuild/jni/AIOUSB_DAC.h, 637
DACOutputClose
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputCloseNoEnd
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputFrame
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputFrameRaw
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputOpen
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputSetCount
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputSetInterlock
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACOutputStart
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 635
ndkbuild/jni/AIOUSB_DAC.h, 638
DACRange
AIOTypes.h, 525
ndkbuild/jni/AIOTypes.h, 547
DACSetBoardRange
AIOUSB_DAC.c, 633
AIOUSB_DAC.h, 636
libaiousb.h, 743
ndkbuild/jni/AIOUSB_DAC.c, 634
ndkbuild/jni/AIOUSB_DAC.h, 637
DACsUsed
AIOUSBDevice, 166
dCBWDataTransferLength
command_block_wrapper, 198
dCBWSignature
command_block_wrapper, 198
dCBWTag
command_block_wrapper, 198
dCSWDataResidue
command_status_wrapper, 199
dCSWSignature
command_status_wrapper, 199
dCSWTag
command_status_wrapper, 199
DEBUG
TestCaseSetup.h, 852
DEBUG_LEVEL
TestCaseSetup.h, 851
DEF_CAL_CHANNEL
TestCaseSetup.h, 852
DEF_MAX_CHANNELS
TestCaseSetup.h, 852
DEF_NUM_CHANNELS
TestCaseSetup.h, 852
DEFAULT_VISIBILITY
android/config.h, 754
msvc/config.h, 755
Xcode/config.h, 755
DESC_HEADER_LENGTH
descriptor.c, 771
DEV_PASS
windows_winusb.c, 817
DEVICE_CTX
libusbi.h, 783
DEVICE_DESC_LENGTH
descriptor.c, 771
libusbi.h, 782
DEVICE_SPEED
windows_winusb.h, 826
DEVICES_REQUIRED
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
write_sample.c, 876
DEVINST
windows_winusb.h, 826
DEVNODE
windows_winusb.h, 826
DEVPATH
openbsd_usb.c, 796
DIO_ConfigurationQuery
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_Configure
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642

libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ConfigureEx
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ConfigureWithDIOBuf
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 640
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_Read1
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_Read8
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ReadAll
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ReadAllToCharStr
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ReadAllToDIOBuf
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_ReadIntoDIOBuf
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_StreamClose
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_StreamFrame
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 646
DIO_StreamOpen
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_StreamSetClocks
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 644
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_Write1
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 712
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_Write8
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIO_WriteAll
AIOUSB_DIO.c, 639
AIOUSB_DIO.h, 642
libaiousb.h, 711
ndkbuild/jni/AIOUSB_DIO.c, 641
ndkbuild/jni/AIOUSB_DIO.h, 645
DIOBuf, 224
buffer, 225
size, 225
strbuf, 225
strbuf_size, 225
DIOBuf.c
DIOBufByteSize, 683
DIOBufGetByteAtIndex, 684
DIOBufGetIndex, 684
DIOBufReplaceBinString, 683
DIOBufReplaceString, 683
DIOBufResize, 683
DIOBufSetByteAtIndex, 684
DIOBufSetIndex, 684
DIOBufSize, 683
DIOBufToBinary, 684
DIOBufToHex, 683
DIOBufToInvertedBinary, 684
DIOBufToString, 683
DeleteDIOBuf, 683
NewDIOBuf, 682
NewDIOBufFromBinStr, 683
NewDIOBufFromChar, 683
DIOBuf.h
DIOBufByteSize, 689
DIOBufGetByteAtIndex, 690
DIOBufGetIndex, 690
DIOBufReplaceBinString, 689
DIOBufReplaceString, 689
DIOBufResize, 689
DIOBufSetByteAtIndex, 690
DIOBufSetIndex, 690
DIOBufSize, 689
DIOBufToBinary, 689
DIOBufToHex, 689
DIOBufToInvertedBinary, 689
DIOBufToString, 689
DIOBufferType, 688
DeleteDIOBuf, 689
NewDIOBuf, 688
NewDIOBufFromBinStr, 688
NewDIOBufFromChar, 688
DIOBufByteSize
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufGetByteAtIndex
DIOBuf.c, 684
DIOBuf.h, 690

libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 687
ndkbuild/jni/DIOBuf.h, 693
DIOBufGetIndex
DIOBuf.c, 684
DIOBuf.h, 690
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 687
ndkbuild/jni/DIOBuf.h, 693
DIOBufReplaceBinString
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufReplaceString
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufResize
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufSetByteAtIndex
DIOBuf.c, 684
DIOBuf.h, 690
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 687
ndkbuild/jni/DIOBuf.h, 693
DIOBufSetIndex
DIOBuf.c, 684
DIOBuf.h, 690
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 687
ndkbuild/jni/DIOBuf.h, 693
DIOBufSize
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufToBinary
DIOBuf.c, 684
DIOBuf.h, 689
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufToHex
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufToInvertedBinary
DIOBuf.c, 684
DIOBuf.h, 689
libaiousb.h, 710
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufToString
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 711
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DIOBufferType
DIOBuf.h, 688
ndkbuild/jni/DIOBuf.h, 691
DIOBytes
AIODeviceInfo, 161
AIOUSBDevice, 166
DIOConfigBits
AIOUSBDevice, 166
DIOPorts
DeviceProperties, 217
DIOStreamSubsystem, 225
AIOUSB::DIOStreamSubsystem, 226
AIOUSB::USBDeviceBase, 318
DLL_DECLARE_FUNC
wince_usb.h, 808, 809
windows_common.h, 810
windows_winusb.h, 830, 831
DLL_DECLARE_HANDLE
wince_usb.h, 808
windows_common.h, 810
windows_winusb.h, 830, 831
DLL_FREE_HANDLE
windows_common.h, 810
DLL_GET_HANDLE
windows_common.h, 810
DLL_LOAD_FUNC
windows_common.h, 811
DLL_LOAD_LIBRARY
windows_common.h, 810
DLL_STRINGIFY
windows_common.h, 810
dRef
AIOUSB_ADC.c, 569
ndkbuild/jni/AIOUSB_ADC.c, 584
DUMMY_HANDLE
poll_windows.h, 799
DUMP
AIOCommandLine.h, 404
aiocommon.c, 849
ndkbuild/jni/AIOCommandLine.h, 407
dac
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_DA12_8A_Family, 291
AIOUSB::USB_DA12_8E_Family, 294
daisample.c
main, 867
daitest.cpp
exit_sample, 856
handle_signal, 856
main, 856
old_action, 856
darwin_backend
darwin_usb.c, 786
libusbi.h, 785
darwin_cached_device, 213
active_config, 214
address, 214
can_enumerate, 214
dev_descriptor, 213
device, 214
first_config, 214
list, 213
location, 213
open_count, 214
parent_session, 213
port, 214
refcount, 214
session, 213
sys_path, 214
darwin_device_handle_priv, 214
cfSource, 214
interfaces, 214
is_open, 214
darwin_device_handle_priv::darwin_interface, 215

cfSource, 215
endpoint_addrs, 215
frames, 215
interface, 215
num_endpoints, 215
darwin_device_priv, 214
dev, 215
darwin_transfer_priv, 215
isoc_framelist, 215
num_iso_packets, 215
req, 215
result, 215
size, 215
darwin_usb.c
darwin_backend, 786
darwin_usb.h
IO_OBJECT_NULL, 787
io_cf_plugin_ref_t, 787
io_notification_port_t, 787
data
USB_CONFIGURATION_DESCRIPTOR_SHORT, 287
usb_raw_command, 310
usbfs_bulktransfer, 326
usbfs_ctrltransfer, 327
usbfs_ioctl, 328
dataBuf
TestCaseSetup, 275
datafd
endpoint, 230
ddesc
device_priv, 216
debug
ADCConfigBlock, 146
AIOArgument, 155
AIOContinuousBuf, 161
configuration, 200
libusb_context, 239
USBDevice, 313
debug_fixed
libusb_context, 239
debug_level
AIOCommandLineOptions, 158
opts, 264
default_aiobuf_json
AIOCommandLineOptions, 159
opts, 264
default_clock_rate
AIOCommandLineOptions, 158
opts, 264
default_end_channel
AIOCommandLineOptions, 159
opts, 264
default_num_channels
AIOCommandLineOptions, 158
opts, 263
default_num_oversamples
AIOCommandLineOptions, 158
opts, 264
default_num_scans
AIOCommandLineOptions, 158
opts, 263
default_out
AIOCountsConverter.c, 451
ndkbuild/jni/AIOCountsConverter.c, 453
default_start_channel
AIOCommandLineOptions, 159
opts, 264
defined
AIOBuf, 155
mux_settings, 260
DeleteADCConfigBlock
ADCConfigBlock.c, 360
ADCConfigBlock.h, 371
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 364
ndkbuild/jni/ADCConfigBlock.h, 374
DeleteAIOBuf
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
DeleteAIOChannelMask
AIOChannelMask.c, 380
AIOChannelMask.h, 385
libaiousb.h, 723
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 388
DeleteAIOChannelRange
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
DeleteAIOCmd
AIOCmd.c, 395
AIOCmd.h, 396
ndkbuild/jni/AIOCmd.c, 395
ndkbuild/jni/AIOCmd.h, 396
DeleteAIOCommandLineOptions
AIOCommandLine.c, 398
AIOCommandLine.h, 405
libaiousb.h, 747
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 409
DeleteAIOConfiguration
AIOConfiguration.c, 410
ndkbuild/jni/AIOConfiguration.c, 411
DeleteAIOContinuousBuf
AIOContinuousBuffer.c, 419
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 445
DeleteAIOCountsConverter
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
DeleteAIODeviceInfo
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
DeleteAIODeviceQuery
AIODeviceQuery.c, 460
AIODeviceQuery.h, 465
libaiousb.h, 727
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 468
DeleteAIOFifo
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
DeleteAIOFifoTYPE
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
DeleteAIOGainRange
AIOCountsConverter.c, 452

AIOCountsConverter.h, 454
libaiousb.h, 727
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 456
DeleteAIOProductGroup
AIOProductTypes.c, 505
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
DeleteAIOProductRange
AIOProductTypes.c, 504
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
DeleteAIOUSBDevice
AIOUSBDevice.c, 666
ndkbuild/jni/AIOUSBDevice.c, 667
DeleteCStringArray
CStringArray.c, 680
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
DeleteDIOBuf
DIOBuf.c, 683
DIOBuf.h, 689
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 686
ndkbuild/jni/DIOBuf.h, 692
DeleteUSBDevice
libaiousb.h, 745
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 751
USBDevice.c, 750
USBDevice.h, 753
DeleteUSBDevices
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 753
DeleteWDGConfig
AIOUSB_WDG.c, 659
AIOUSB_WDG.h, 663
libaiousb.h, 745
ndkbuild/jni/AIOUSB_WDG.c, 661
ndkbuild/jni/AIOUSB_WDG.h, 664
Deleteint
AIOList.c, 499
ndkbuild/jni/AIOList.c, 500
Deleteintlist
AIOList.c, 500
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
delta
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
deprecated/classlib/AI16_DataPoint.cpp, 343
deprecated/classlib/AI16_DataPoint.hpp, 343
deprecated/classlib/AI16_DataSet.cpp, 343
deprecated/classlib/AI16_DataSet.hpp, 343
deprecated/classlib/AI16_InputRange.cpp, 344
deprecated/classlib/AI16_InputRange.hpp, 344
deprecated/classlib/AO16_AnalogOutputSubsystem.cpp, 346
deprecated/classlib/AO16_AnalogOutputSubsystem.hpp, 346
deprecated/classlib/AO16_OutputRange.cpp, 346
deprecated/classlib/AO16_OutputRange.hpp, 346
deprecated/classlib/AnalogIORange.cpp, 345
deprecated/classlib/AnalogIORange.hpp, 345
deprecated/classlib/AnalogInputSubsystem.cpp, 344
deprecated/classlib/AnalogInputSubsystem.hpp, 345
deprecated/classlib/AnalogOutputSubsystem.cpp, 345
deprecated/classlib/AnalogOutputSubsystem.hpp, 346
deprecated/classlib/Counter.cpp, 347
deprecated/classlib/Counter.hpp, 347
deprecated/classlib/CounterSubsystem.cpp, 348
deprecated/classlib/CounterSubsystem.hpp, 348
deprecated/classlib/CppCommon.h, 348
deprecated/classlib/DA12_AnalogOutputSubsystem.cpp, 348
deprecated/classlib/DA12_AnalogOutputSubsystem.hpp, 349
deprecated/classlib/DA12_OutputRange.cpp, 349
deprecated/classlib/DA12_OutputRange.hpp, 349
deprecated/classlib/DIOStreamSubsystem.cpp, 351
deprecated/classlib/DIOStreamSubsystem.hpp, 351
deprecated/classlib/DeviceSubsystem.cpp, 350
deprecated/classlib/DeviceSubsystem.hpp, 350
deprecated/classlib/DigitalIOSubsystem.cpp, 350
deprecated/classlib/DigitalIOSubsystem.hpp, 350
deprecated/classlib/OutputVoltagePoint.hpp, 351
deprecated/classlib/README.doc, 352
deprecated/classlib/USB_AI16_Family.cpp, 352
deprecated/classlib/USB_AI16_Family.hpp, 352
deprecated/classlib/USB_AIO16_Family.cpp, 353
deprecated/classlib/USB_AIO16_Family.hpp, 353
deprecated/classlib/USB_AO16_Family.cpp, 353
deprecated/classlib/USB_AO16_Family.hpp, 353
deprecated/classlib/USB_CTR_15_Family.cpp, 354
deprecated/classlib/USB_CTR_15_Family.hpp, 354
deprecated/classlib/USB_DA12_8A_Family.cpp, 354
deprecated/classlib/USB_DA12_8A_Family.hpp, 354
deprecated/classlib/USB_DA12_8E_Family.cpp, 355
deprecated/classlib/USB_DA12_8E_Family.hpp, 355
deprecated/classlib/USB_DIO_16_Family.cpp, 355
deprecated/classlib/USB_DIO_16_Family.hpp, 355
deprecated/classlib/USB_DIO_32_Family.cpp, 356
deprecated/classlib/USB_DIO_32_Family.hpp, 356
deprecated/classlib/USB_DIO_Family.cpp, 356
deprecated/classlib/USB_DIO_Family.hpp, 356
deprecated/classlib/USBDeviceBase.cpp, 357
deprecated/classlib/USBDeviceBase.hpp, 357
deprecated/classlib/USBDeviceManager.cpp, 358
deprecated/classlib/USBDeviceManager.hpp, 358
depth
windows_device_priv, 338
desc
wince_device_priv, 337
describe
libusb_version, 258
Descriptor
USBDevice, 313
descriptor
usb_raw_command, 310
descriptor.c
CONFIG_DESC_LENGTH, 771
DESC_HEADER_LENGTH, 771
DEVICE_DESC_LENGTH, 771
usbi_device_cache_descriptor, 771
usbi_get_config_index_by_value, 771
usbi_parse_descriptor, 771
descriptors
linux_device_priv, 259
descriptors_len
linux_device_priv, 259
designation
driver_lookup, 229
fx_known_device, 231
windows_usb_api_backend, 339
destroy_device
usbi_os_backend, 332
detach_kernel_driver

usbi_os_backend, 332
dev
aiousb_libusb_args, 164
darwin_device_priv, 215
libusb_device_handle, 244
wince_device_priv, 337
dev_capability
libusb_bos_descriptor, 235
dev_capability_data
libusb_bos_dev_capability_descriptor, 236
dev_class
libusb_hotplug_callback, 247
dev_descr
sunos_device_priv, 269
dev_descriptor
darwin_cached_device, 213
windows_device_priv, 338
dev_handle
interface_handle_t, 235
libusb_transfer, 255
dev_mem_alloc
usbi_os_backend, 332
dev_mem_free
usbi_os_backend, 332
device
ADCConfigBlock, 145
darwin_cached_device, 214
libusb_hotplug_message, 248
ram_poke_context, 267
usb_raw_command, 310
USBDevice, 313
Device handling and enumeration, 92
LIBUSB_BM_LPM_SUPPORT, 95
LIBUSB_BM_LTM_SUPPORT, 95
LIBUSB_BT_CONTAINER_ID, 95
LIBUSB_BT_SS_USB_DEVICE_CAPABILITY, 95
LIBUSB_BT_USB_2_0_EXTENSION, 95
LIBUSB_BT_WIRELESS_USB_DEVICE_CAPABILITY, 95
LIBUSB_FULL_SPEED_OPERATION, 95
LIBUSB_HIGH_SPEED_OPERATION, 95
LIBUSB_LOW_SPEED_OPERATION, 95
LIBUSB_SPEED_FULL, 95
LIBUSB_SPEED_HIGH, 95
LIBUSB_SPEED_LOW, 95
LIBUSB_SPEED_SUPER, 95
LIBUSB_SPEED_UNKNOWN, 95
LIBUSB_SUPER_SPEED_OPERATION, 95
libusb_attach_kernel_driver, 104
libusb_bos_type, 95
libusb_claim_interface, 100
libusb_clear_halt, 101
libusb_close, 99
libusb_detach_kernel_driver, 103
libusb_device, 94
libusb_device_handle, 94
libusb_free_device_list, 96
libusb_get_bus_number, 96
libusb_get_configuration, 99
libusb_get_device, 99
libusb_get_device_address, 97
libusb_get_device_list, 96
libusb_get_device_speed, 97
libusb_get_max_iso_packet_size, 98
libusb_get_max_packet_size, 97
libusb_get_parent, 97
libusb_get_port_number, 96
libusb_get_port_numbers, 96
libusb_get_port_path, 97
libusb_kernel_driver_active, 103
libusb_open, 98
libusb_open_device_with_vid_pid, 99
libusb_ref_device, 98
libusb_release_interface, 101
libusb_reset_device, 103
libusb_set_auto_detach_kernel_driver, 104
libusb_set_configuration, 100
libusb_set_interface_alt_setting, 101
libusb_speed, 95
libusb_ss_usb_device_capability_attributes, 95
libusb_supported_speed, 95
libusb_unref_device, 98
libusb_usb_2_0_extension_attributes, 95
Device hotplug event notification, 118
LIBUSB_HOTPLUG_ENUMERATE, 119
LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED, 119
LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT, 119
LIBUSB_HOTPLUG_NO_FLAGS, 119
libusb_hotplug_callback_fn, 118
libusb_hotplug_callback_handle, 118
libusb_hotplug_deregister_callback, 120
libusb_hotplug_event, 119
libusb_hotplug_flag, 119
libusb_hotplug_register_callback, 119
DeviceCausedOvercurrent
windows_winusb.h, 830
DeviceConnected
windows_winusb.h, 829
DeviceFailedEnumeration
windows_winusb.h, 829
DeviceGeneralFailure
windows_winusb.h, 830
DeviceHubNestedTooDeeply
windows_winusb.h, 830
DeviceInLegacyHub
windows_winusb.h, 830
DeviceNotEnoughBandwidth
windows_winusb.h, 830
DeviceNotEnoughPower
windows_winusb.h, 830
device_address
libusb_device, 242
device_args
AIOArguments, 155
device_close
libusb_context, 240
device_descriptor
libusb_device, 242
device_handle_priv_size
usbi_os_backend, 333
device_index
configuration, 200
device_priv, 216
cdesc, 216
ddesc, 216
devname, 216
devnode, 216
fd, 216
device_priv_size
usbi_os_backend, 333
DeviceAddress
USB_NODE_CONNECTION_INFORMATION_EX, 306
deviceDesc
aiousb_libusb_args, 164
USBDevice, 313
DeviceDescriptor
AIOUSBDevice.h, 668
ndkbuild/jni/AIOUSBDevice.h, 669
tag_USB_DK_DEVICE_INFO, 270
USB_NODE_CONNECTION_INFORMATION_EX, 306
DeviceEnum

AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
deviceFound
TestCaseSetup, 275
deviceHandle
USBDevice, 313
DeviceID
tag_USB_DK_DEVICE_ID, 270
DeviceIndex
AIOContinuousBuf, 160
BulkAcquireWorkerParams, 197
TestCaseSetup, 275
deviceIndex
AIOUSB::USBDeviceBase, 318
AIOUSBDevice, 166
DeviceInfo, 216
index, 216
name, 216
nameSize, 216
numCounters, 216
numDIOBytes, 216
outputMask, 216
productID, 216
readBuffer, 216
serialNumber, 216
writeBuffer, 216
DeviceIsHub
USB_NODE_CONNECTION_INFORMATION_EX, 306
deviceList
AIOUSB::USBDeviceManager, 325
DeviceProperties, 217
ADCChannels, 218
ADCChannelsPerGroup, 218
ADCMUXChannels, 218
Counters, 217
DACChannels, 218
DIOPorts, 217
Name, 217
ProductID, 217
RootClock, 217
SerialNumber, 217
Tristates, 217
DeviceSubsystem, 218
AIOUSB::DeviceSubsystem, 218
deviceTable
AIODeviceTable.c, 473
AIODeviceTable.h, 480
ndkbuild/jni/AIODeviceTable.c, 476
ndkbuild/jni/AIODeviceTable.h, 484
DeviceTableAtIndex
AIOUSB_Core.c, 610
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 620
DeviceTableAtIndex_Lock
AIOUSB_Core.c, 611
AIOUSB_Core.h, 617
libaiousb.h, 713
ndkbuild/jni/AIOUSB_Core.c, 613
ndkbuild/jni/AIOUSB_Core.h, 621
devices
discovered_devs, 229
devlink_cbarg, 219
minor, 219
myself, 219
nargs, 219
devname
device_priv, 216
devnode
device_priv, 216
devnum
usbfs_connectinfo, 326
diFirst
AIOTypes.h, 525
ndkbuild/jni/AIOTypes.h, 547
diNone
AIOTypes.h, 525
ndkbuild/jni/AIOTypes.h, 547
diOnly
AIOTypes.h, 525
ndkbuild/jni/AIOTypes.h, 547
differentialMode
AIOUSB::AI16_DataPoint, 148
AIOUSB::AnalogInputSubsystem, 187
DigitalIOSubsystem, 219
AIOUSB::DigitalIOSubsystem, 220
digitalIOSubsystem
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 300
AIOUSB::USB_DIO_Family, 302
dio
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 302
dio96_read_write.c
BITS_PER_BYTE, 871
DEVICES_REQUIRED, 871
find_dio_96, 872
MAKE_MASK, 871
MASK_BYTES, 871
MAX_DIO_BYTES, 871
MAX_NAME_SIZE, 871
main, 872
PORT_A, 871
PORT_B, 871
PORT_C, 871
show_byte, 872
dio_sample.c
find_ai_board, 856
main, 856
dioStreamSubsystem
AIOUSB::USB_DIO_16_Family, 297
diostream
AIOUSB::USB_DIO_16_Family, 297
diotest.c
main, 857
diotest2.cpp
main, 857
direction
mock_usb_xfers.c, 698
dirktest.c
capture_data, 857
fnd, 857
fp, 858
get_channel_range, 857
main, 858
process_cmd_line, 857
discard_first_sample
configuration, 200
discardFirstSample
ADCConfigBlock, 146
aio_counts_converter, 154
AIOUSB::AI16_DataSet, 151
AIOUSBDevice, 166
discdevs
node_args, 261

discovered_devs, 229
capacity, 229
devices, 229
len, 229
discovered_devs_append
core.c, 769
libusbi.h, 785
dlink_hdl
node_args, 261
Do
USBTransfer, 335
doBulkAcquire
TestCaseSetup, 274
doBulkConfigBlock
TestCaseSetup, 273
doCSVReadVoltages
TestCaseSetup, 274
doCSVWithGetChannelV
TestCaseSetup, 274
doCleanupAfterBulk
TestCaseSetup, 274
doDACDirect
TestCaseSetup, 274
doDACDirectSetup
TestCaseSetup, 274
doDemonstrateReadVoltages
TestCaseSetup, 274
doDisplayBulkResults
TestCaseSetup, 275
doFastITScan
TestCaseSetup, 274
doFastITScanSetup
TestCaseSetup, 274
doGenericVendorWrite
TestCaseSetup, 274
doGetBuffer
TestCaseSetup, 274
DoLoadCalTable
AIOUSB_ADC.c, 567
ndkbuild/jni/AIOUSB_ADC.c, 581
doPreReadImmediateVoltages
TestCaseSetup, 274
doPreSetup
TestCaseSetup, 273
doScanSingleChannel
TestCaseSetup, 274
doSetAutoCalibration
TestCaseSetup, 273
doSomething
AIOUSB_WDG.c, 659
ndkbuild/jni/AIOUSB_WDG.c, 661
TestCaseSetup, 273
doTestSetAutoCalibration
TestCaseSetup, 274
doVerifyGroundCalibration
TestCaseSetup, 273
doVerifyReferenceCalibration
TestCaseSetup, 274
doc/aiousb.doc, 359
doc/firmware.doc, 359
doc/index.doc, 359
doc/install.doc, 359
doc/java.doc, 359
doc/libusb.doc, 359
doc/samples.doc, 359
doc/wrappers.doc, 359
done
hotplugtest.c, 764
DoubleArray, 229
AIOUSB::DoubleArray, 229
dpfp.c
MODE_AWAIT_FINGER_OFF, 756
MODE_AWAIT_FINGER_ON, 756
MODE_CAPTURE, 756
MODE_INIT, 756
MODE_READY, 756
MODE_SHUT_UP, 756
STATE_AWAIT_IMAGE, 757
STATE_AWAIT_IRQ_FINGER_DETECTED, 757
STATE_AWAIT_IRQ_FINGER_REMOVED, 757
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGE-R_OFF, 757
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGE-R_ON, 757
STATE_AWAIT_MODE_CHANGE_CAPTURE, 757
dpfp.c
CTRL_IN, 756
CTRL_OUT, 756
EP_DATA, 756
EP_INTR, 756
INTR_LENGTH, 756
main, 757
USB_RQ, 756
dpfp_threaded.c
MODE_AWAIT_FINGER_OFF, 758
MODE_AWAIT_FINGER_ON, 758
MODE_CAPTURE, 758
MODE_INIT, 758
MODE_READY, 758
MODE_SHUT_UP, 758
STATE_AWAIT_IMAGE, 758
STATE_AWAIT_IRQ_FINGER_DETECTED, 758
STATE_AWAIT_IRQ_FINGER_REMOVED, 758
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGE-R_OFF, 758
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGE-R_ON, 758
STATE_AWAIT_MODE_CHANGE_CAPTURE, 758
dpfp_threaded.c
CTRL_IN, 757
CTRL_OUT, 757
EP_DATA, 757
EP_INTR, 757
INTR_LENGTH, 757
main, 758
SEM_NAME, 758
USB_RQ, 757
dpriv
sunos_device_handle_priv, 268
driver
usbfs_disconnect_claim, 327
usbfs_getdriver, 327
driver_lookup, 229
designation, 229
list, 229
reg_prop, 229
driver_name_list
windows_usb_api_backend, 339

E2BIG
errno.h, 833
EEPROM_CUSTOM_BASE_ADDRESS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
EEPROM_CUSTOM_MAX_ADDRESS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
EEPROM_CUSTOM_MIN_ADDRESS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
EEPROM_SERIAL_NUMBER_ADDRESS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
END

AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
END_CHANNEL
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
ERROR
linux_usbfs.c, 792
EACCES
errno.h, 834
EAGAIN
errno.h, 834
EBADF
errno.h, 833
EBUSY
errno.h, 834
ECHILD
errno.h, 833
EDEADLK
errno.h, 834
EDEADLOCK
errno.h, 834
EDOM
errno.h, 834
EEXIST
errno.h, 834
EFAULT
errno.h, 834
EFBIG
errno.h, 834
EINTR
errno.h, 833
EINVAL
errno.h, 834
EIO
errno.h, 833
EISDIR
errno.h, 834
EMFILE
errno.h, 834
EMLINK
errno.h, 834
ENABLE_LOGGING
android/config.h, 754
msvc/config.h, 755
Xcode/config.h, 755
ENFILE
errno.h, 834
ENODEV
errno.h, 834
ENOENT
errno.h, 833
ENOEXEC
errno.h, 833
ENOFILE
errno.h, 833
ENOMEM
errno.h, 834
ENOSPC
errno.h, 834
ENOTDIR
errno.h, 834
ENOTTY
errno.h, 834
ENXIO
errno.h, 833
EP_DATA
dpfp.c, 756
dpfp_threaded.c, 757
EP_DATA_IN
sam3u_benchmark.c, 765
EP_INTR
dpfp.c, 756
dpfp_threaded.c, 757
EP_ISO_IN
sam3u_benchmark.c, 765
EPERM
errno.h, 833
EPIPE
errno.h, 834
ERANGE
errno.h, 834
EROFS
errno.h, 834
ERR_BUFFER_SIZE
windows_common.h, 810
ERR_EXIT
xusb.c, 766
ERROR
TestCaseSetup.h, 852
ERROR_LEVEL
TestCaseSetup.h, 851
ESPIPE
errno.h, 834
ESRCH
errno.h, 833
ETIMEDOUT
threads_windows.h, 805
EXDEV
errno.h, 834
EXIT_CODE
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 876
EXPORTED_FUNCTION
AIOTypes.h, 517
ndkbuild/jni/AIOTypes.h, 539
emptyDeviceList
AIOUSB::USBDeviceManager, 321
end
aio_channel_range, 153
end_channel
AIOChannelRangeTmp, 157
AIOCommandLineOptions, 159
channel_range, 197
opts, 264
endChannel
AIOUSB::AnalogInputSubsystem, 187
endpoint, 230
datafd, 230
libusb_interface_descriptor, 250
libusb_transfer, 255
statfd, 230
usb_raw_command, 310, 311
usbfs_urb, 330
windows_device_priv, 338
endpoint_addrs
darwin_device_handle_priv::darwin_interface, 215
endpoint_etc
usb_raw_command, 310
endpoint_index
usb_raw_command, 310
EndpointAddress
tag_USB_DK_TRANSFER_REQUEST, 271
EndpointDescriptor
USB_PIPE_INFO, 308
EndpointToIndex
USBDevice, 313
EndpointToInterface
USBDevice, 313
endpoints
handle_priv, 231
endpos
AIOBuf, 155
enhanced_out
AIOCountsConverter.c, 451

ndkbuild/jni/AIOCountsConverter.c, 453
EntryCreated
WatchedEntry, 336
EntryRemoved
WatchedEntry, 336
EnumStringLookup
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
envGetDouble
TestCaseSetup, 274
envGetInteger
TestCaseSetup, 274
ep
usbfs_bulktransfer, 326
epoch_time
wince_usb.c, 806
windows_nt_common.c, 812
eps
sunos_device_handle_priv, 268
usbfs_streams, 329
errmsg
aio_ret_value, 154
errno
errno.h, 834
errno.h
E2BIG, 833
EACCES, 834
EAGAIN, 834
EBADF, 833
EBUSY, 834
ECHILD, 833
EDEADLK, 834
EDEADLOCK, 834
EDOM, 834
EEXIST, 834
EFAULT, 834
EFBIG, 834
EINTR, 833
EINVAL, 834
EIO, 833
EISDIR, 834
EMFILE, 834
EMLINK, 834
ENFILE, 834
ENODEV, 834
ENOENT, 833
ENOEXEC, 833
ENOFILE, 833
ENOMEM, 834
ENOSPC, 834
ENOTDIR, 834
ENOTTY, 834
ENXIO, 833
EPERM, 833
EPIPE, 834
ERANGE, 834
EROFS, 834
ESPIPE, 834
ESRCH, 833
EXDEV, 834
errno, 834
Error, 230
Error, 230
message, 230
what, 230
error_count
usbfs_urb, 330
error_message
aioerror, 162
event
libusb_hotplug_message, 248
timer_request, 276
usbi_cond_perthread, 331
event_data_lock
libusb_context, 240
event_flags
libusb_context, 240
event_handler_active
libusb_context, 240
event_handling_key
libusb_context, 240
event_pipe
libusb_context, 239
event_waiters_cond
libusb_context, 240
event_waiters_lock
libusb_context, 240
events
libusb_hotplug_callback, 247
libusb_pollfd, 251
pollfd, 266
events_lock
libusb_context, 240
exit
usbi_os_backend, 332
windows_usb_api_backend, 339
exit_polling
poll_windows.c, 797
poll_windows.h, 800
exit_sample
daitest.cpp, 856
exitcode
AIOContinuousBuf, 161
extcal.cpp
main, 861
extra
libusb_config_descriptor, 238
libusb_endpoint_descriptor, 245
libusb_interface_descriptor, 250
extra_length
libusb_config_descriptor, 238
libusb_endpoint_descriptor, 246
libusb_interface_descriptor, 250
ezusb.c
internal_only, 759
skip_external, 759
skip_internal, 759
ezusb.c
ezusb_load_ram, 759
logerror, 759
RETRY_LIMIT, 759
RW_INTERNAL, 759
RW_MEMORY, 759
ram_mode, 759
verbose, 759
ezusb.h
ezusb_load_eeprom, 760
ezusb_load_ram, 760
FX_KNOWN_DEVICES, 760
FX_TYPE_AN21, 760
FX_TYPE_FX1, 760
FX_TYPE_FX2, 760
FX_TYPE_FX2LP, 760
FX_TYPE_FX3, 760
FX_TYPE_MAX, 760
FX_TYPE_NAMES, 760
FX_TYPE_UNDEFINED, 760
IMG_TYPE_BIX, 760
IMG_TYPE_HEX, 760
IMG_TYPE_IIC, 760
IMG_TYPE_IMG, 760
IMG_TYPE_MAX, 760
IMG_TYPE_NAMES, 760
IMG_TYPE_UNDEFINED, 760

verbose, 760
ezusb_load_eeprom
ezusb.h, 760
ezusb_load_ram
ezusb.c, 759
ezusb.h, 760

FIFO_Method_begin
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
FIFO_Method_end
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
FILE_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
FACILITY_SETUPAPI
windows_nt_common.h, 813
FATAL
TestCaseSetup.h, 852
FATAL_LEVEL
TestCaseSetup.h, 851
fActiveConfiguration
USBDevice, 314
fCancelled
USBTransfer, 335
fClaimedInterfaces
USBDevice, 314
USBDeviceHandle, 319
fConfigToIndex
USBDevice, 314
fConfigurationDescriptors
USBDevice, 314
fDevice
WatchedEntry, 336
fDeviceDescriptor
USBDevice, 314
fEndpointToIndex
USBDevice, 314
fEndpointToInterface
USBDevice, 314
fEntries
WatchedEntry, 336
FIFO_Method
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
FILE_ANY_ACCESS
windows_winusb.h, 825
FILE_DEVICE_USB
windows_winusb.h, 825
FIRMWARE
fxload.c, 761
FIRST_ENUM
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
fInitCheck
RosterLooper, 268
USBDevice, 314
USBDeviceHandle, 319
WatchedEntry, 336
fIsDirectory
WatchedEntry, 336
fLibusbTransfer
USBTransfer, 335
fLink
WatchedEntry, 336
fLooper
USBRoster, 334
fMessenger
RosterLooper, 268
WatchedEntry, 336
fNode
WatchedEntry, 336
fPath
USBDevice, 314
fRawFD
USBDeviceHandle, 319
fRoot
RosterLooper, 267
fRoster
RosterLooper, 267
fStatusLock
USBTransfer, 335
fTransfers
USBDeviceHandle, 319
fTransfersLock
USBDeviceHandle, 319
fTransfersSem
USBDeviceHandle, 319
fTransfersThread
USBDeviceHandle, 319
fUSBDevice
USBDeviceHandle, 319
USBTransfer, 335
fUsbiTransfer
USBTransfer, 335
FX_KNOWN_DEVICES
ezusb.h, 760
FX_TYPE_AN21
ezusb.h, 760
FX_TYPE_FX1
ezusb.h, 760
FX_TYPE_FX2
ezusb.h, 760
FX_TYPE_FX2LP
ezusb.h, 760
FX_TYPE_FX3
ezusb.h, 760
FX_TYPE_MAX
ezusb.h, 760
FX_TYPE_NAMES
ezusb.h, 760
FX_TYPE_UNDEFINED
ezusb.h, 760
FXLOAD_VERSION
fxload.c, 761
false
libusb_testlib.h, 846
windows_common.h, 809
xusb.c, 766
FastITBakConfig
AIOUSBDevice, 168
FastITConfig
AIOUSBDevice, 168
FastITConfig_size
AIOUSBDevice, 168
fd
device_priv, 216
libusb_pollfd, 251
linux_device_handle_priv, 258
pollfd, 266
winfd, 340
fd_added_cb
libusb_context, 240
fd_cb_user_data
libusb_context, 240
fd_removed
linux_device_handle_priv, 258
fd_removed_cb
libusb_context, 240
fd_to_winfd
poll_windows.c, 797
poll_windows.h, 800
feature_report_size

hid_device_priv, 231
FeatureReportByteLength
HIDP_CAPS, 233
fifo
AIOContinuousBuf, 160
file_handle
configuration, 200
file_name
configuration, 200
FilterID
tag_USB_DK_DEVICE_INFO, 270
find_ai_board
burst_test.c, 854
dio_sample.c, 856
jni/read_channels_test.c, 861
julian_test.c, 858
read_channels_test.c, 862
sample_dio.c, 859
stream_test.c, 866
find_dio
read_and_write.c, 871
find_dio_96
dio96_read_write.c, 872
mytest.c, 873
tmp.c, 875
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 877
find_idio
iiro_sample.c, 880
findDevice
TestCaseSetup, 273
Firmware/README.doc, 352
first_config
darwin_cached_device, 214
flag
option, 262
flags
libusb_hotplug_callback, 247
libusb_transfer, 255
usbfs_disconnect_claim, 327
usbfs_urb, 330
FlashSectors
AIOUSBDevice, 167
FlushPipe
winusb_interface, 341
flying_transfers
libusb_context, 240
flying_transfers_lock
libusb_context, 240
fnd
continuous_mode.c, 854
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 855
dirktest.c, 857
simple_continuous_with_json.c, 865
start_stop_continuous.c, 865
test.c, 860
write_sample.c, 877
foo.c
foo_something, 848
foo.h
foo_something, 848
foo_something
foo.c, 848
foo.h, 848
foreach_CStringArray_p
AIOList.h, 502
ndkbuild/jni/AIOList.h, 503
foreach_array
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
foreach_int
AIOList.h, 502
ndkbuild/jni/AIOList.h, 503
fp
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 856
dirktest.c, 858
simple_continuous_with_json.c, 865
start_stop_continuous.c, 865
test.c, 860
framePoints
config_options, 199
frames
darwin_device_handle_priv::darwin_interface, 215
Free
winusb_interface, 341
free_fn
cJSON_Hooks, 198
free_streams
usbi_os_backend, 332
FullSpeed
windows_usbdk.h, 815
FullSpeed
windows_winusb.h, 826
function
libusb_testlib_test, 254
fx_known_device, 230
designation, 231
pid, 231
type, 231
vid, 231
fxload.c
ARRAYSIZE, 761
FIRMWARE, 761
FXLOAD_VERSION, 761
LOADER, 761
logerror, 761
main, 761

GAIN
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
GENERIC_DOSOMETHING_PLACEHOLDER
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
G_BREAKPOINT
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
G_STMT_END
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
G_STMT_START
AIOTypes.h, 519
ndkbuild/jni/AIOTypes.h, 541
GCC_VERSION
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
GEN_PASS
windows_winusb.c, 817
GET_ENDPOINT
AIOUSB_DIO.c, 638
ndkbuild/jni/AIOUSB_DIO.c, 640
gInitCount
haiku_usb_raw.cpp, 788
GRAB_RESOURCE
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 497
GREEN
AIOUSB_Log.h, 647
ndkbuild/jni/AIOUSB_Log.h, 649
gUsbRoster
haiku_usb_raw.cpp, 788
gain

ad_gain_pairs, 145
aio_channel_range, 153
gain_code
AIOCommandLineOptions, 158
opts, 264
gain_ranges
aio_counts_converter, 153
gainCodes
TestCaseSetup, 275
gaincode
AIOChannelRangeTmp, 157
channel_range, 197
GenResult
tag_USB_DK_TRANSFER_RESULT, 272
generic
usb_raw_command, 310
generic_etc
usb_raw_command, 310
generic_index
usb_raw_command, 310
GenericVendorRead
AIOUSB_Core.c, 612
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 615
ndkbuild/jni/AIOUSB_Core.h, 623
GenericVendorWrite
AIOUSB_Core.c, 612
AIOUSB_Core.h, 618
libaiousb.h, 715
ndkbuild/jni/AIOUSB_Core.c, 614
ndkbuild/jni/AIOUSB_Core.h, 623
get_active_config_descriptor
usbi_os_backend, 332
get_cal_mode
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 365
get_channel_range
aiocommon.c, 849
aiocommon.h, 850
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 855
dirktest.c, 857
simple_continuous_with_json.c, 864
start_stop_continuous.c, 865
test.c, 860
get_config_descriptor
usbi_os_backend, 332
get_config_descriptor_by_value
usbi_os_backend, 332
get_configuration
usbi_os_backend, 332
get_device_descriptor
usbi_os_backend, 332
get_device_list
usbi_os_backend, 332
get_gain_code
ADCConfigBlock.c, 362
ndkbuild/jni/ADCConfigBlock.c, 365
get_options
read_channels_with_getchannelv_test.cpp, 862
get_usb_device
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 754
getAIOUSBVersion
AIOUSB::USBDeviceManager, 321
getAIOUSBVersionDate
AIOUSB::USBDeviceManager, 321
GetAssociatedInterface
winusb_interface, 341
getCalMode
AIOUSB::AI16_DataSet, 150
AIOUSB::AnalogInputSubsystem, 175
getChannel
AIOUSB::AI16_DataPoint, 147
getChannelsPerGroup
AIOUSB::AnalogInputSubsystem, 172
getClock
AIOUSB::AnalogInputSubsystem, 180
AIOUSB::DIOStreamSubsystem, 226
getCommTimeout
AIOUSB::USBDeviceBase, 316
getConfiguration
AIOUSB::DigitalIOSubsystem, 222
getCounter
AIOUSB::CounterSubsystem, 205
getCounts
AIOUSB::AI16_DataPoint, 147
TestCaseSetup, 275
GetCurrentAlternateSetting
winusb_interface, 341
GetDescriptor
winusb_interface, 341
getDeviceByProductID
AIOUSB::USBDeviceManager, 324, 325
GetDeviceBySerialNumber
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 655
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 653
ndkbuild/jni/AIOUSB_Properties.h, 657
getDeviceBySerialNumber
AIOUSB::USBDeviceManager, 325
getDeviceIndex
AIOUSB::Counter, 201
AIOUSB::DeviceSubsystem, 218
AIOUSB::USBDeviceBase, 316
GetDeviceSerialNumber
AIOUSB_Properties.c, 652
AIOUSB_Properties.h, 655
libaiousb.h, 744
ndkbuild/jni/AIOUSB_Properties.c, 653
ndkbuild/jni/AIOUSB_Properties.h, 657
GetDevices
AIODeviceTable.c, 471
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 475
ndkbuild/jni/AIODeviceTable.h, 482
getGainCodes
TestCaseSetup, 275
GetHiRef
AIOUSB_ADC.c, 566
ndkbuild/jni/AIOUSB_ADC.c, 581
GetJSONValueOrDefault
AIOContinuousBuffer.c, 424
ndkbuild/jni/AIOContinuousBuffer.c, 434
getMiscClock
AIOUSB::USBDeviceBase, 315
getName
AIOUSB::USBDeviceBase, 316
getNumChannels
AIOUSB::AnalogInputSubsystem, 172
AIOUSB::AnalogOutputSubsystem, 191
AIOUSB::DigitalIOSubsystem, 221
getNumCounterBlocks
AIOUSB::CounterSubsystem, 205
getNumCounters
AIOUSB::CounterSubsystem, 205
getNumMUXChannels
AIOUSB::AnalogInputSubsystem, 172

getNumPorts
AIOUSB::DigitalIOSubsystem, 221
getNumTristateChannels
AIOUSB::DigitalIOSubsystem, 221
getNumTristateGroups
AIOUSB::DigitalIOSubsystem, 221
getOverSample
AIOUSB::AI16_DataSet, 150
AIOUSB::AnalogInputSubsystem, 179
GetOverlappedResult
winusb_interface, 341
getParent
AIOUSB::DeviceSubsystem, 219
GetPipePolicy
winusb_interface, 341
getPoints
AIOUSB::AI16_DataSet, 150
GetPowerPolicy
winusb_interface, 341
getProductID
AIOUSB::USBDeviceBase, 316
getRange
AIOUSB::AI16_DataPoint, 147
AIOUSB::AnalogInputSubsystem, 176
AIOUSB::AnalogIORange, 189
AIOUSB::AO16_AnalogOutputSubsystem, 193
AIOUSB::DA12_AnalogOutputSubsystem, 209
getRangeText
AIOUSB::AI16_DataPoint, 147
AIOUSB::AnalogInputSubsystem, 173
AIOUSB::AO16_AnalogOutputSubsystem, 193
AIOUSB::DA12_AnalogOutputSubsystem, 208
getResultCodeAsString
AIOUSB::USBDeviceManager, 323
GetSafeDeviceName
AIODeviceTable.c, 472
AIODeviceTable.h, 479
libaiousb.h, 730
ndkbuild/jni/AIODeviceTable.c, 475
ndkbuild/jni/AIODeviceTable.h, 482
getSerialNumber
AIOUSB::USBDeviceBase, 316
getStreamingBlockSize
AIOUSB::AnalogInputSubsystem, 180
AIOUSB::DIOStreamSubsystem, 226
AIOUSB::USBDeviceBase, 316
getSubsystem
AIOUSB::AI16_DataSet, 150
getSupportedProductIDs
AIOUSB::USB_AI16_Family, 279
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 284
AIOUSB::USB_CTR_15_Family, 288
AIOUSB::USB_DA12_8A_Family, 290
AIOUSB::USB_DA12_8E_Family, 292
AIOUSB::USB_DIO_16_Family, 296
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 301
getSupportedProductNames
AIOUSB::USB_AI16_Family, 279
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 284
AIOUSB::USB_CTR_15_Family, 288
AIOUSB::USB_DA12_8A_Family, 290
AIOUSB::USB_DA12_8E_Family, 292
AIOUSB::USB_DIO_16_Family, 296
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 301
getTimeStamp
AIOUSB::AI16_DataSet, 150
getTriggerMode
AIOUSB::AI16_DataSet, 150
AIOUSB::AnalogInputSubsystem, 175
getVolts
AIOUSB::AI16_DataPoint, 148
TestCaseSetup, 275
getenv
getopt.c, 762
missing.c, 841
missing.h, 841
getopt
getopt.c, 762
getopt.h, 763
getopt.c
PERMUTE, 762
REQUIRE_ORDER, 762
RETURN_IN_ORDER, 762
getopt.c
const, 762
getenv, 762
getopt, 762
NONOPTION_P, 762
optarg, 762
opterr, 762
optind, 762
optopt, 762
SWAP_FLAGS, 762
getopt.h
getopt, 763
getopt_long, 763
getopt_long_only, 763
no_argument, 763
optarg, 763
opterr, 763
optind, 763
optional_argument, 763
optopt, 763
required_argument, 763
getopt1.c
const, 763
getopt_long, 763
getopt_long_only, 763
NULL, 763
getopt_long
getopt.h, 763
getopt1.c, 763
getopt_long_only
getopt.h, 763
getopt1.c, 763
goDoIt
HOLD/reverse_cal_table.cpp, 858
reverse_cal_table.cpp, 859
groupcpy
AIOProductTypes.c, 505
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509

HID_REPORT_TYPE_FEATURE
windows_winusb.h, 829
HID_REPORT_TYPE_INPUT
windows_winusb.h, 829
HID_REPORT_TYPE_OUTPUT
windows_winusb.h, 829
HID_REQ_GET_IDLE
windows_winusb.h, 829
HID_REQ_GET_PROTOCOL
windows_winusb.h, 829
HID_REQ_GET_REPORT
windows_winusb.h, 829
HID_REQ_SET_IDLE
windows_winusb.h, 829
HID_REQ_SET_PROTOCOL
windows_winusb.h, 829

HID_REQ_SET_REPORT
windows_winusb.h, 829
HANDLE_CTX
libusbi.h, 784
HANDLE_VALID
windows_winusb.c, 816
HAS_PTHREAD
AIOTypes.h, 517
ndkbuild/jni/AIOTypes.h, 539
HAVE_ASM_TYPES_H
android/config.h, 755
HAVE_DLFCN_H
android/config.h, 754
HAVE_GETTIMEOFDAY
android/config.h, 754
Xcode/config.h, 755
HAVE_INTTYPES_H
android/config.h, 754
HAVE_MEMORY_H
android/config.h, 754
HAVE_POLL_H
android/config.h, 754
Xcode/config.h, 755
HAVE_SIGNAL_H
android/config.h, 754
msvc/config.h, 755
HAVE_SYS_SOCKET_H
android/config.h, 755
HAVE_SYS_STAT_H
android/config.h, 754
HAVE_SYS_TIME_H
android/config.h, 754
Xcode/config.h, 755
HAVE_SYS_TYPES_H
android/config.h, 754
msvc/config.h, 755
HAVE_UNISTD_H
android/config.h, 754
HCD_PASS
windows_winusb.c, 817
hDACDataMutex
AIOUSBDevice, 167
hDACDataSem
AIOUSBDevice, 167
HID_CTL_CODE
windows_winusb.h, 824
HID_GET_IDLE
xusb.c, 766
HID_GET_PROTOCOL
xusb.c, 766
HID_GET_REPORT
xusb.c, 766
HID_IN_CTL_CODE
windows_winusb.h, 824
HID_IN_EP
windows_winusb.h, 824
HID_OUT_CTL_CODE
windows_winusb.h, 824
HID_OUT_EP
windows_winusb.h, 824
HID_PASS
windows_winusb.c, 817
HID_SET_IDLE
xusb.c, 766
HID_SET_PROTOCOL
xusb.c, 766
HID_SET_REPORT
xusb.c, 766
HIDD_ATTRIBUTES, 232
ProductID, 232
Size, 232
VendorID, 232
VersionNumber, 232
HIDP_CAPS, 232
FeatureReportByteLength, 233
InputReportByteLength, 233
NumberFeatureButtonCaps, 233
NumberFeatureDataIndices, 233
NumberFeatureValueCaps, 233
NumberInputButtonCaps, 233
NumberInputDataIndices, 233
NumberInputValueCaps, 233
NumberLinkCollectionNodes, 233
NumberOutputButtonCaps, 233
NumberOutputDataIndices, 233
NumberOutputValueCaps, 233
OutputReportByteLength, 233
Reserved, 233
Usage, 233
UsagePage, 233
HIDP_REPORT_TYPE
windows_winusb.h, 829
HIDP_VALUE_CAPS
windows_winusb.h, 829
HOLD/reverse_cal_table.cpp
goDoIt, 858
main, 858
HTAB_SIZE
windows_nt_common.c, 812
HUB_PASS
windows_winusb.c, 817
haiku_usb_raw.h
B_USB_RAW_COMMAND_BULK_TRANSFER, 789
B_USB_RAW_COMMAND_CLEAR_FEATURE, 789
B_USB_RAW_COMMAND_CONTROL_TRANSFER, 789
B_USB_RAW_COMMAND_GET_ACTIVE_ALT_INTERFACE_INDEX, 789
B_USB_RAW_COMMAND_GET_ALT_INTERFACE_COUNT, 789
B_USB_RAW_COMMAND_GET_CONFIGURATION_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_DEVICE_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_ENDPOINT_DESCRIPTOR_ETC, 789
B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_GENERIC_DESCRIPTOR_ETC, 789
B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_INTERFACE_DESCRIPTOR_ETC, 789
B_USB_RAW_COMMAND_GET_STATUS, 789
B_USB_RAW_COMMAND_GET_STRING_DESCRIPTOR, 789
B_USB_RAW_COMMAND_GET_VERSION, 789
B_USB_RAW_COMMAND_INTERRUPT_TRANSFER, 789
B_USB_RAW_COMMAND_ISOCHRONOUS_TRANSFER, 789
B_USB_RAW_COMMAND_SET_ALT_INTERFACE, 789
B_USB_RAW_COMMAND_SET_CONFIGURATION, 789
B_USB_RAW_COMMAND_SET_FEATURE, 789
B_USB_RAW_STATUS_ABORTED, 789
B_USB_RAW_STATUS_CRC_ERROR, 789
B_USB_RAW_STATUS_FAILED, 789

B_USB_RAW_STATUS_INVALID_CONFIGURATION, 789
B_USB_RAW_STATUS_INVALID_ENDPOINT, 789
B_USB_RAW_STATUS_INVALID_INTERFACE, 789
B_USB_RAW_STATUS_INVALID_STRING, 789
B_USB_RAW_STATUS_NO_MEMORY, 789
B_USB_RAW_STATUS_STALLED, 789
B_USB_RAW_STATUS_SUCCESS, 789
B_USB_RAW_STATUS_TIMEOUT, 789
haiku_usb_raw.cpp
gInitCount, 788
gUsbRoster, 788
haiku_usb_raw_backend, 788
haiku_usb_raw.h
usb_raw_command_id, 789
usb_raw_command_status, 789
haiku_usb_raw_backend
haiku_usb_raw.cpp, 788
libusbi.h, 786
handle
aiousb_libusb_args, 164
hotplugtest.c, 764
libusb_hotplug_callback, 247
winfd, 340
handle_events
usbi_os_backend, 332
handle_priv, 231
endpoints, 231
handle_signal
daitest.cpp, 856
handle_to_winfd
poll_windows.c, 798
poll_windows.h, 800
handle_transfer_completion
usbi_os_backend, 332
has_arg
option, 262
HasOverlappedIoCompletedSync
poll_windows.h, 799
hid
windows_device_priv, 338
hid_buffer
windows_transfer_priv, 339
hid_dest
windows_transfer_priv, 339
hid_device_priv, 231
config, 231
feature_report_size, 231
input_report_size, 231
nb_interfaces, 231
output_report_size, 231
pid, 231
string, 231
string_index, 232
uses_report_ids, 231
vid, 231
hid_expected_size
windows_transfer_priv, 339
HighSpeed
windows_usbdk.h, 815
HighSpeed
windows_winusb.h, 826
hotplug.c
usbi_hotplug_deregister_all, 771
usbi_hotplug_match, 771
usbi_hotplug_notification, 771
hotplug.h
libusb_hotplug_callback, 772
libusb_hotplug_message, 772
usbi_hotplug_deregister_all, 772
usbi_hotplug_match, 772
usbi_hotplug_notification, 772
hotplug_cbs
libusb_context, 240
hotplug_cbs_lock
libusb_context, 240
hotplug_msgs
libusb_context, 240
hotplug_poll
usbi_os_backend, 332
hotplugtest.c
done, 764
handle, 764
main, 764
htab_entry, 233
str, 233
used, 233
windows_nt_common.c, 812
htab_hash
windows_nt_common.c, 812
windows_nt_common.h, 813
HubDescriptor
USB_HUB_INFORMATION, 304
HubInformation
USB_NODE_INFORMATION, 307
HubIs2xCapable
USB_HUB_CAPABILITIES, 303
HubIsArmedWakeOnConnect
USB_HUB_CAP_FLAGS, 303
HubIsBusPowered
USB_HUB_INFORMATION, 304
HubIsHighSpeed
USB_HUB_CAP_FLAGS, 303
HubIsHighSpeedCapable
USB_HUB_CAP_FLAGS, 303
HubIsMultiTt
USB_HUB_CAP_FLAGS, 303
HubIsMultiTtCapable
USB_HUB_CAP_FLAGS, 303
HubIsRoot
USB_HUB_CAP_FLAGS, 303
hz
AIOContinuousBuf, 160

IN
mock_capture_usb.c, 695
mock_dio.c, 697
mock_usb_xfers.c, 698
INDEX_NUM
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
INVALID_OBJECT
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
iConfiguration
libusb_config_descriptor, 237
USB_CONFIGURATION_DESCRIPTOR, 286
ID
tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST, 269
tag_USB_DK_DEVICE_INFO, 270
IGNORE_RETVAL
testlib.c, 848
iInterface
libusb_interface_descriptor, 250
USB_INTERFACE_DESCRIPTOR, 305
IMG_TYPE_BIX
ezusb.h, 760
IMG_TYPE_HEX
ezusb.h, 760
IMG_TYPE_IIC
ezusb.h, 760
IMG_TYPE_IMG

ezusb.h, 760
IMG_TYPE_MAX
ezusb.h, 760
IMG_TYPE_NAMES
ezusb.h, 760
IMG_TYPE_UNDEFINED
ezusb.h, 760
iManufacturer
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 278
INFO
TestCaseSetup.h, 852
INFO_LEVEL
TestCaseSetup.h, 851
INPUT_TYPE
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 498
INQUIRY_LENGTH
xusb.c, 766
INT16_C
stdint.h, 844
INT16_MAX
stdint.h, 843
INT16_MIN
stdint.h, 843
INT32_C
stdint.h, 844
INT32_MAX
stdint.h, 843
INT32_MIN
stdint.h, 843
INT64_C
stdint.h, 844
INT64_MAX
stdint.h, 843
INT64_MIN
stdint.h, 843
INT8_C
stdint.h, 844
INT8_MAX
stdint.h, 843
INT8_MIN
stdint.h, 843
INT_FAST16_MAX
stdint.h, 844
INT_FAST16_MIN
stdint.h, 843
INT_FAST32_MAX
stdint.h, 844
INT_FAST32_MIN
stdint.h, 843
INT_FAST64_MAX
stdint.h, 844
INT_FAST64_MIN
stdint.h, 843
INT_FAST8_MAX
stdint.h, 844
INT_FAST8_MIN
stdint.h, 843
INT_LEAST16_MAX
stdint.h, 843
INT_LEAST16_MIN
stdint.h, 843
INT_LEAST32_MAX
stdint.h, 843
INT_LEAST32_MIN
stdint.h, 843
INT_LEAST64_MAX
stdint.h, 843
INT_LEAST64_MIN
stdint.h, 843
INT_LEAST8_MAX
stdint.h, 843
INT_LEAST8_MIN
stdint.h, 843
INTERFACE_CLAIMED
windows_winusb.h, 823
INTERNAL_METHOD
ndkbuild/jni/USBDevice.h, 751
USBDevice.h, 753
INTMAX_C
stdint.h, 844
INTMAX_MAX
stdint.h, 844
INTMAX_MIN
stdint.h, 844
INTPTR_MAX
stdint.h, 844
INTPTR_MIN
stdint.h, 844
INTR_LENGTH
dpfp.c, 756
dpfp_threaded.c, 757
INVALID_FD
testlib.c, 848
INVALID_WINFD
poll_windows.c, 798
poll_windows.h, 800
IO_DIRECTION
mock_capture_usb.c, 695
mock_dio.c, 697
mock_usb_xfers.c, 698
IO_OBJECT_NULL
darwin_usb.h, 787
IOCTL_USBFS_BULK
linux_usbfs.h, 794
IOCTL_USBFS_IOCTL
linux_usbfs.h, 794
IOCTL_USBFS_RESET
linux_usbfs.h, 794
iProduct
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 278
IS_EPIN
libusbi.h, 784
IS_EPOUT
libusbi.h, 784
IS_XFERIN
libusbi.h, 784
IS_XFEROUT
libusbi.h, 784
iSerialNumber
libusb_device_descriptor, 244
UKW_DEVICE_DESCRIPTOR, 278
ITRANSFER_CTX
libusbi.h, 784
id
ProductIDName, 266
windows_usb_api_backend, 339
idProduct
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
idVendor
libusb_device_descriptor, 243
UKW_DEVICE_DESCRIPTOR, 277
idio_sample.c
main, 878
RATE_LIMIT, 878
idio_sample2.c
main, 879
RATE_LIMIT, 878
iface
USBDevice, 313
ifno

usbfs_ioctl, 328
iiro_sample.c
find_idio, 880
main, 880
RATE_LIMIT, 880
IllegalArgumentException, 233
AIOUSB::IllegalArgumentException, 234
imaxdiv_t, 234
quot, 234
rem, 234
ImmADCs
AIOUSBDevice, 167
ImmDACs
AIOUSBDevice, 166
increment
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
Index
tag_USB_DK_CONFIG_DESCRIPTOR_REQUEST, 269
index
AIOCommandLineOptions, 159
AIODeviceQuery, 162
DeviceInfo, 216
opts, 264
usb_raw_command, 310
WINUSB_SETUP_PACKET, 342
init
usbi_os_backend, 332
windows_usb_api_backend, 339
init_device
mock_capture_usb.c, 695
init_polling
poll_windows.c, 797
poll_windows.h, 800
InitCheck
RosterLooper, 267
USBDevice, 313
USBDeviceHandle, 319
WatchedEntry, 336
Initialise
USBDevice, 313
Initialize
winusb_interface, 341
initialize
AIOUSB::USB_AI16_Family, 279
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 284
AIOUSB::USB_CTR_15_Family, 288
AIOUSB::USB_DA12_8A_Family, 290
AIOUSB::USB_DA12_8E_Family, 292
AIOUSB::USB_DIO_16_Family, 296
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 301
InitializeUSBDevice
libaiousb.h, 745
mock_capture_usb.c, 696
ndkbuild/jni/USBDevice.c, 748
ndkbuild/jni/USBDevice.h, 751
USBDevice.c, 750
USBDevice.h, 753
initialized
winusb_interface, 341
input_report_size
hid_device_priv, 231
inputRange
AIOUSB::AnalogInputSubsystem, 187
InputReportByteLength
HIDP_CAPS, 233
installation, 5
InstanceID
tag_USB_DK_DEVICE_ID, 270
int16_t
stdint.h, 845
int32_t
stdint.h, 845
int64_t
stdint.h, 845
int8_t
stdint.h, 845
int_fast16_t
stdint.h, 845
int_fast32_t
stdint.h, 845
int_fast64_t
stdint.h, 845
int_fast8_t
stdint.h, 845
int_least16_t
stdint.h, 845
int_least32_t
stdint.h, 845
int_least64_t
stdint.h, 845
int_least8_t
stdint.h, 845
IntArray, 234
AIOUSB::IntArray, 234
intToString
AIOList.c, 499
ndkbuild/jni/AIOList.c, 500
interface
darwin_device_handle_priv::darwin_interface, 215
libusb_config_descriptor, 238
usb_raw_command, 310, 311
usbfs_disconnect_claim, 327
usbfs_getdriver, 327
usbfs_setinterface, 329
interface_etc
usb_raw_command, 310
interface_handle
windows_device_handle_priv, 337
interface_handle_t, 234
api_handle, 235
dev_handle, 235
interface_index
usb_raw_command, 310
interface_number
wince_transfer_priv, 337
windows_transfer_priv, 339
interfaces
darwin_device_handle_priv, 214
internal_only
ezusb.c, 759
IntertuptTransferType
windows_usbdk.h, 816
Interval
WINUSB_PIPE_INFORMATION, 342
intlistFirst
AIOList.c, 499
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
intlistInsert
AIOList.c, 500
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
intlistSize
AIOList.c, 500
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
intlistToString

AIOList.c, 500
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
intmax_t
stdint.h, 845
intptr_t
stdint.h, 844
inttypes.h
PRIX16, 838
PRIX32, 838
PRIX64, 838
PRIX8, 838
PRIXFAST16, 839
PRIXFAST32, 839
PRIXFAST64, 839
PRIXFAST8, 839
PRIXLEAST16, 839
PRIXLEAST32, 839
PRIXLEAST64, 839
PRIXLEAST8, 838
PRIXMAX, 839
PRIXPTR, 840
PRId16, 837
PRId32, 837
PRId64, 837
PRId8, 837
PRIdFAST16, 837
PRIdFAST32, 837
PRIdFAST64, 837
PRIdFAST8, 837
PRIdLEAST16, 837
PRIdLEAST32, 837
PRIdLEAST64, 837
PRIdLEAST8, 837
PRIdMAX, 837
PRIdPTR, 840
PRIi16, 837
PRIi32, 837
PRIi64, 837
PRIi8, 837
PRIiFAST16, 837
PRIiFAST32, 837
PRIiFAST64, 837
PRIiFAST8, 837
PRIiLEAST16, 837
PRIiLEAST32, 837
PRIiLEAST64, 837
PRIiLEAST8, 837
PRIiMAX, 837
PRIiPTR, 840
PRIo16, 837
PRIo32, 837
PRIo64, 837
PRIo8, 837
PRIoFAST16, 838
PRIoFAST32, 838
PRIoFAST64, 838
PRIoFAST8, 837
PRIoLEAST16, 837
PRIoLEAST32, 837
PRIoLEAST64, 837
PRIoLEAST8, 837
PRIoMAX, 838
PRIoPTR, 840
PRIu16, 838
PRIu32, 838
PRIu64, 838
PRIu8, 838
PRIuFAST16, 838
PRIuFAST32, 838
PRIuFAST64, 838
PRIuFAST8, 838
PRIuLEAST16, 838
PRIuLEAST32, 838
PRIuLEAST64, 838
PRIuLEAST8, 838
PRIuMAX, 838
PRIuPTR, 840
PRIx16, 838
PRIx32, 838
PRIx64, 838
PRIx8, 838
PRIxFAST16, 838
PRIxFAST32, 838
PRIxFAST64, 838
PRIxFAST8, 838
PRIxLEAST16, 838
PRIxLEAST32, 838
PRIxLEAST64, 838
PRIxLEAST8, 838
PRIxMAX, 838
PRIxPTR, 840
SCNd16, 839
SCNd32, 839
SCNd64, 839
SCNdFAST16, 839
SCNdFAST32, 839
SCNdFAST64, 839
SCNdLEAST16, 839
SCNdLEAST32, 839
SCNdLEAST64, 839
SCNdMAX, 839
SCNdPTR, 840
SCNi16, 839
SCNi32, 839
SCNi64, 839
SCNiFAST16, 839
SCNiFAST32, 839
SCNiFAST64, 839
SCNiLEAST16, 839
SCNiLEAST32, 839
SCNiLEAST64, 839
SCNiMAX, 839
SCNiPTR, 840
SCNo16, 839
SCNo32, 839
SCNo64, 839
SCNoFAST16, 839
SCNoFAST32, 840
SCNoFAST64, 840
SCNoLEAST16, 839
SCNoLEAST32, 839
SCNoLEAST64, 839
SCNoMAX, 840
SCNoPTR, 840
SCNu16, 840
SCNu32, 840
SCNu64, 840
SCNuFAST16, 840
SCNuFAST32, 840
SCNuFAST64, 840
SCNuLEAST16, 840
SCNuLEAST32, 840
SCNuLEAST64, 840
SCNuMAX, 840
SCNuPTR, 840
SCNx16, 840
SCNx32, 840
SCNx64, 840
SCNxFAST16, 840
SCNxFAST32, 840
SCNxFAST64, 840
SCNxLEAST16, 840

SCNxLEAST32, 840
SCNxLEAST64, 840
SCNxMAX, 840
SCNxPTR, 840
io.c
usbi_add_pollfd, 773
usbi_handle_disconnect, 773
usbi_handle_transfer_cancellation, 773
usbi_handle_transfer_completion, 773
usbi_io_exit, 773
usbi_io_init, 773
usbi_remove_pollfd, 773
usbi_signal_transfer_completion, 773
io_cf_plugin_ref_t
darwin_usb.h, 787
io_notification_port_t
darwin_usb.h, 787
ioctl_code
usbfs_ioctl, 328
ipollfds
libusb_context, 240
is_all_digits
ADCConfigBlock.c, 362
ADCConfigBlock.h, 371
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 366
ndkbuild/jni/ADCConfigBlock.h, 374
is_open
darwin_device_handle_priv, 214
is_polling_set
poll_windows.c, 798
isAutoCalPresent
AIOUSB::AnalogInputSubsystem, 172
isAutoConfig
AIOUSB::AnalogInputSubsystem, 173
IsCancelled
USBTransfer, 335
isDifferentialMode
AIOUSB::AI16_DataPoint, 147
AIOUSB::AnalogInputSubsystem, 177
isDiscardFirstSample
AIOUSB::AI16_DataSet, 150
AIOUSB::AnalogInputSubsystem, 174
isInit
AIOUSBDevice, 166
isOpen
AIOUSB::USBDeviceManager, 323
isSupportedProductID
AIOUSB::USB_AI16_Family, 279
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_CTR_15_Family, 288
AIOUSB::USB_DA12_8A_Family, 290
AIOUSB::USB_DA12_8E_Family, 292
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 301
iso_frame_desc
usbfs_urb, 330
iso_packet_desc
libusb_transfer, 256
iso_packet_offset
linux_transfer_priv, 259
iso_urbs
linux_transfer_priv, 259
isoc_framelist
darwin_transfer_priv, 215
isochronous
usb_raw_command, 311
IsochronousTransferType
windows_usbdk.h, 816
IsochronousPacketsArray
tag_USB_DK_TRANSFER_REQUEST, 271
IsochronousPacketsArraySize
tag_USB_DK_TRANSFER_REQUEST, 271
IsochronousResultsArray
tag_USB_DK_TRANSFER_RESULT, 272
itransfer
winfd, 340

JOINED
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
JSON
AIOUSB_Properties.h, 655
ndkbuild/jni/AIOUSB_Properties.h, 656, 657
JCONF
AIOCommandLine.h, 404
aiocommon.c, 849
ndkbuild/jni/AIOCommandLine.h, 407
jni/read_channels_test.c
find_ai_board, 861
main, 861
julian_test.c
find_ai_board, 858
main, 858

KEEP
mock_usb_xfers.c, 698
KLIB_VERSION
windows_winusb.h, 829
KUSB_FNID
windows_winusb.h, 829
kernel_driver_active
usbi_os_backend, 332

L
AIOWDGConfig, 168
LIBUSB_BM_LPM_SUPPORT
Device handling and enumeration, 95
LIBUSB_BM_LTM_SUPPORT
Device handling and enumeration, 95
LIBUSB_BT_CONTAINER_ID
Device handling and enumeration, 95
LIBUSB_BT_SS_USB_DEVICE_CAPABILITY
Device handling and enumeration, 95
LIBUSB_BT_USB_2_0_EXTENSION
Device handling and enumeration, 95
LIBUSB_BT_WIRELESS_USB_DEVICE_CAPABILITY
Device handling and enumeration, 95
LIBUSB_CAP_HAS_CAPABILITY
Miscellaneous, 107
LIBUSB_CAP_HAS_HID_ACCESS
Miscellaneous, 107
LIBUSB_CAP_HAS_HOTPLUG
Miscellaneous, 107
LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER
Miscellaneous, 107
LIBUSB_CLASS_APPLICATION
USB descriptors, 112
LIBUSB_CLASS_AUDIO
USB descriptors, 111
LIBUSB_CLASS_COMM
USB descriptors, 111
LIBUSB_CLASS_CONTENT_SECURITY
USB descriptors, 112
LIBUSB_CLASS_DATA
USB descriptors, 112
LIBUSB_CLASS_DIAGNOSTIC_DEVICE
USB descriptors, 112
LIBUSB_CLASS_HID
USB descriptors, 111
LIBUSB_CLASS_HUB
USB descriptors, 112

LIBUSB_CLASS_IMAGE
USB descriptors, 112
LIBUSB_CLASS_MASS_STORAGE
USB descriptors, 112
LIBUSB_CLASS_PER_INTERFACE
USB descriptors, 111
LIBUSB_CLASS_PERSONAL_HEALTHCARE
USB descriptors, 112
LIBUSB_CLASS_PHYSICAL
USB descriptors, 111
LIBUSB_CLASS_PRINTER
USB descriptors, 112
LIBUSB_CLASS_PTP
USB descriptors, 112
LIBUSB_CLASS_SMART_CARD
USB descriptors, 112
LIBUSB_CLASS_VENDOR_SPEC
USB descriptors, 112
LIBUSB_CLASS_VIDEO
USB descriptors, 112
LIBUSB_CLASS_WIRELESS
USB descriptors, 112
LIBUSB_DT_BOS
USB descriptors, 112
LIBUSB_DT_CONFIG
USB descriptors, 112
LIBUSB_DT_DEVICE
USB descriptors, 112
LIBUSB_DT_DEVICE_CAPABILITY
USB descriptors, 112
LIBUSB_DT_ENDPOINT
USB descriptors, 112
LIBUSB_DT_HID
USB descriptors, 112
LIBUSB_DT_HUB
USB descriptors, 112
LIBUSB_DT_INTERFACE
USB descriptors, 112
LIBUSB_DT_PHYSICAL
USB descriptors, 112
LIBUSB_DT_REPORT
USB descriptors, 112
LIBUSB_DT_SS_ENDPOINT_COMPANION
USB descriptors, 112
LIBUSB_DT_STRING
USB descriptors, 112
LIBUSB_DT_SUPERSPEED_HUB
USB descriptors, 112
LIBUSB_ENDPOINT_IN
USB descriptors, 112
LIBUSB_ENDPOINT_OUT
USB descriptors, 112
LIBUSB_ERROR_ACCESS
Miscellaneous, 107
LIBUSB_ERROR_BUSY
Miscellaneous, 107
LIBUSB_ERROR_INTERRUPTED
Miscellaneous, 107
LIBUSB_ERROR_INVALID_PARAM
Miscellaneous, 107
LIBUSB_ERROR_IO
Miscellaneous, 107
LIBUSB_ERROR_NO_DEVICE
Miscellaneous, 107
LIBUSB_ERROR_NO_MEM
Miscellaneous, 107
LIBUSB_ERROR_NOT_FOUND
Miscellaneous, 107
LIBUSB_ERROR_NOT_SUPPORTED
Miscellaneous, 107
LIBUSB_ERROR_OTHER
Miscellaneous, 107
LIBUSB_ERROR_OVERFLOW
Miscellaneous, 107
LIBUSB_ERROR_PIPE
Miscellaneous, 107
LIBUSB_ERROR_TIMEOUT
Miscellaneous, 107
LIBUSB_FULL_SPEED_OPERATION
Device handling and enumeration, 95
LIBUSB_HIGH_SPEED_OPERATION
Device handling and enumeration, 95
LIBUSB_HOTPLUG_ENUMERATE
Device hotplug event notification, 119
LIBUSB_HOTPLUG_EVENT_DEVICE_ARRIVED
Device hotplug event notification, 119
LIBUSB_HOTPLUG_EVENT_DEVICE_LEFT
Device hotplug event notification, 119
LIBUSB_HOTPLUG_NO_FLAGS
Device hotplug event notification, 119
LIBUSB_ISO_SYNC_TYPE_ADAPTIVE
USB descriptors, 113
LIBUSB_ISO_SYNC_TYPE_ASYNC
USB descriptors, 113
LIBUSB_ISO_SYNC_TYPE_NONE
USB descriptors, 113
LIBUSB_ISO_SYNC_TYPE_SYNC
USB descriptors, 113
LIBUSB_ISO_USAGE_TYPE_DATA
USB descriptors, 113
LIBUSB_ISO_USAGE_TYPE_FEEDBACK
USB descriptors, 113
LIBUSB_ISO_USAGE_TYPE_IMPLICIT
USB descriptors, 113
LIBUSB_LOG_LEVEL_DEBUG
Library initialization/deinitialization, 90
LIBUSB_LOG_LEVEL_ERROR
Library initialization/deinitialization, 90
LIBUSB_LOG_LEVEL_INFO
Library initialization/deinitialization, 90
LIBUSB_LOG_LEVEL_NONE
Library initialization/deinitialization, 90
LIBUSB_LOG_LEVEL_WARNING
Library initialization/deinitialization, 90
LIBUSB_LOW_SPEED_OPERATION
Device handling and enumeration, 95
LIBUSB_RECIPIENT_DEVICE
Miscellaneous, 107
LIBUSB_RECIPIENT_ENDPOINT
Miscellaneous, 107
LIBUSB_RECIPIENT_INTERFACE
Miscellaneous, 107
LIBUSB_RECIPIENT_OTHER
Miscellaneous, 107
LIBUSB_REQUEST_CLEAR_FEATURE
Miscellaneous, 106
LIBUSB_REQUEST_GET_CONFIGURATION
Miscellaneous, 106
LIBUSB_REQUEST_GET_DESCRIPTOR
Miscellaneous, 106
LIBUSB_REQUEST_GET_INTERFACE
Miscellaneous, 106
LIBUSB_REQUEST_GET_STATUS
Miscellaneous, 106
LIBUSB_REQUEST_SET_ADDRESS
Miscellaneous, 106
LIBUSB_REQUEST_SET_CONFIGURATION
Miscellaneous, 106
LIBUSB_REQUEST_SET_DESCRIPTOR
Miscellaneous, 106
LIBUSB_REQUEST_SET_FEATURE
Miscellaneous, 106
LIBUSB_REQUEST_SET_INTERFACE
Miscellaneous, 106

LIBUSB_REQUEST_SET_SEL
Miscellaneous, 107
LIBUSB_REQUEST_SYNCH_FRAME
Miscellaneous, 106
LIBUSB_REQUEST_TYPE_CLASS
Miscellaneous, 107
LIBUSB_REQUEST_TYPE_RESERVED
Miscellaneous, 107
LIBUSB_REQUEST_TYPE_STANDARD
Miscellaneous, 107
LIBUSB_REQUEST_TYPE_VENDOR
Miscellaneous, 107
LIBUSB_SET_ISOCH_DELAY
Miscellaneous, 107
LIBUSB_SPEED_FULL
Device handling and enumeration, 95
LIBUSB_SPEED_HIGH
Device handling and enumeration, 95
LIBUSB_SPEED_LOW
Device handling and enumeration, 95
LIBUSB_SPEED_SUPER
Device handling and enumeration, 95
LIBUSB_SPEED_UNKNOWN
Device handling and enumeration, 95
LIBUSB_SUCCESS
Miscellaneous, 107
LIBUSB_SUPER_SPEED_OPERATION
Device handling and enumeration, 95
LIBUSB_TRANSFER_ADD_ZERO_PACKET
Asynchronous device I/O, 126
LIBUSB_TRANSFER_CANCELLED
Asynchronous device I/O, 126
LIBUSB_TRANSFER_COMPLETED
Asynchronous device I/O, 126
LIBUSB_TRANSFER_ERROR
Asynchronous device I/O, 126
LIBUSB_TRANSFER_FREE_BUFFER
Asynchronous device I/O, 126
LIBUSB_TRANSFER_FREE_TRANSFER
Asynchronous device I/O, 126
LIBUSB_TRANSFER_NO_DEVICE
Asynchronous device I/O, 126
LIBUSB_TRANSFER_OVERFLOW
Asynchronous device I/O, 126
LIBUSB_TRANSFER_SHORT_NOT_OK
Asynchronous device I/O, 126
LIBUSB_TRANSFER_STALL
Asynchronous device I/O, 126
LIBUSB_TRANSFER_TIMED_OUT
Asynchronous device I/O, 126
LIBUSB_TRANSFER_TYPE_BULK
USB descriptors, 112
LIBUSB_TRANSFER_TYPE_BULK_STREAM
USB descriptors, 112
LIBUSB_TRANSFER_TYPE_CONTROL
USB descriptors, 112
LIBUSB_TRANSFER_TYPE_INTERRUPT
USB descriptors, 112
LIBUSB_TRANSFER_TYPE_ISOCHRONOUS
USB descriptors, 112
LAMBDA
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
LAST_ENUM
AIOTypes.h, 517
ndkbuild/jni/AIOTypes.h, 539
LIBUSB_API_VERSION
Miscellaneous, 106
LIBUSB_CALL
Miscellaneous, 106
LIBUSB_COMPLETED
windows_winusb.h, 824
LIBUSB_DT_BOS_SIZE
libusb.h, 780
LIBUSB_ERROR_COUNT
libusb.h, 780
LIBUSB_MICRO
version.h, 832
LIBUSB_MINOR
version.h, 832
LIBUSB_NANO
version_nano.h, 832
LIBUSB_NULL_TEST
libusb_testlib.h, 846
LIBUSB_RC
version.h, 832
LIBUSB_REQ_IN
windows_winusb.h, 824
LIBUSB_REQ_OUT
windows_winusb.h, 824
LIBUSB_REQ_TYPE
windows_winusb.h, 824
LIBUSBArgs
ndkbuild/jni/USBDevice.h, 751
USBDevice.h, 753
LIBUSBX_API_VERSION
libusb.h, 780
LIST_COUNT
stress.c, 847
LIST_SEPARATOR
windows_winusb.h, 823
LOADER
fxload.c, 761
LOCKING_MECHANISM
AIOFifo, 163
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 497
new_aio_fifo, 261
LOG
TestCaseSetup.h, 851
LOG_LEVEL
AIOUSB_Log.h, 648
ndkbuild/jni/AIOUSB_Log.h, 650
LOOKUP
AIOEither.c, 484
AIOFifo.c, 491
ndkbuild/jni/AIOEither.c, 485
ndkbuild/jni/AIOFifo.c, 493
LOOP_BREAK
windows_winusb.c, 817
LOOP_CHECK
windows_winusb.c, 816
LPOVERLAPPED
poll_windows.c, 798
LPUKW_DEVICE
wince_usb.h, 808
language_id
usb_raw_command, 310
last_ugenpath
node_args, 261
LastDIOData
AIOUSBDevice, 167
left
aio_ret_value, 154
len
discovered_devs, 229
usbfs_bulktransfer, 326
length
libusb_iso_packet_descriptor, 251
libusb_transfer, 256
usb_raw_command, 310, 311
usbfs_iso_packet_desc, 328
WINUSB_SETUP_PACKET, 342
lib/ADCConfigBlock.c, 359

lib/ADCConfigBlock.h, 366
lib/AIOBuf.c, 374
lib/AIOBuf.h, 376
lib/AIOChannelMask.c, 380
lib/AIOChannelMask.h, 384
lib/AIOChannelRange.c, 390
lib/AIOChannelRange.h, 392
lib/AIOCmd.c, 395
lib/AIOCmd.h, 395
lib/AIOCommandLine.c, 396
lib/AIOCommandLine.h, 402
lib/AIOConfiguration.c, 410
lib/AIOConfiguration.h, 411
lib/AIOContinuousBuffer.c, 414
lib/AIOContinuousBuffer.h, 435
lib/AIOCountsConverter.c, 451
lib/AIOCountsConverter.h, 453
lib/AIODeviceInfo.c, 456
lib/AIODeviceInfo.h, 457
lib/AIODeviceQuery.c, 458
lib/AIODeviceQuery.h, 464
lib/AIODeviceTable.c, 469
lib/AIODeviceTable.h, 477
lib/AIOEither.c, 484
lib/AIOEither.h, 486
lib/AIOFifo.c, 490
lib/AIOFifo.h, 494
lib/AIOList.c, 499
lib/AIOList.h, 501
lib/AIOProductTypes.c, 504
lib/AIOProductTypes.h, 506
lib/AIOTuple.c, 510
lib/AIOTuple.h, 510
lib/AIOTypes.h, 512
lib/AIOUSB_ADC.c, 556
lib/AIOUSB_ADC.h, 585
lib/AIOUSB_CTR.c, 623
lib/AIOUSB_CTR.h, 627
lib/AIOUSB_Core.c, 609
lib/AIOUSB_Core.h, 615
lib/AIOUSB_CustomEEPROM.c, 629
lib/AIOUSB_CustomEEPROM.h, 631
lib/AIOUSB_DAC.c, 632
lib/AIOUSB_DAC.h, 635
lib/AIOUSB_DIO.c, 638
lib/AIOUSB_DIO.h, 642
lib/AIOUSB_Log.c, 646
lib/AIOUSB_Log.h, 647
lib/AIOUSB_Properties.c, 651
lib/AIOUSB_Properties.h, 654
lib/AIOUSB_USB.c, 658
lib/AIOUSB_USB.h, 658
lib/AIOUSB_WDG.c, 659
lib/AIOUSB_WDG.h, 662
lib/AIOUSBDevice.c, 665
lib/AIOUSBDevice.h, 667
lib/CStringArray.c, 679
lib/CStringArray.h, 680
lib/DIOBuf.c, 682
lib/DIOBuf.h, 687
lib/USBDevice.c, 749
lib/USBDevice.h, 752
lib/aiousb.h, 556
lib/cJSON.c, 670
lib/cJSON.h, 674
lib/mocks/mock_aiocontbuf_get_data.c, 693
lib/mocks/mock_aiocontbuf_get_data_arduino.c, 694
lib/mocks/mock_capture_usb.c, 694
lib/mocks/mock_dio.c, 696
lib/mocks/mock_usb_xfers.c, 697
lib/ndkbuild/jni/ADCConfigBlock.c, 363
lib/ndkbuild/jni/ADCConfigBlock.h, 371
lib/ndkbuild/jni/AIOBuf.c, 375
lib/ndkbuild/jni/AIOBuf.h, 378
lib/ndkbuild/jni/AIOChannelMask.c, 382
lib/ndkbuild/jni/AIOChannelMask.h, 387
lib/ndkbuild/jni/AIOChannelRange.c, 391
lib/ndkbuild/jni/AIOChannelRange.h, 393
lib/ndkbuild/jni/AIOCmd.c, 395
lib/ndkbuild/jni/AIOCmd.h, 396
lib/ndkbuild/jni/AIOCommandLine.c, 399
lib/ndkbuild/jni/AIOCommandLine.h, 406
lib/ndkbuild/jni/AIOConfiguration.c, 410
lib/ndkbuild/jni/AIOConfiguration.h, 412
lib/ndkbuild/jni/AIOContinuousBuffer.c, 425
lib/ndkbuild/jni/AIOContinuousBuffer.h, 442
lib/ndkbuild/jni/AIOCountsConverter.c, 452
lib/ndkbuild/jni/AIOCountsConverter.h, 454
lib/ndkbuild/jni/AIODeviceInfo.c, 456
lib/ndkbuild/jni/AIODeviceInfo.h, 457
lib/ndkbuild/jni/AIODeviceQuery.c, 461
lib/ndkbuild/jni/AIODeviceQuery.h, 467
lib/ndkbuild/jni/AIODeviceTable.c, 473
lib/ndkbuild/jni/AIODeviceTable.h, 480
lib/ndkbuild/jni/AIOEither.c, 485
lib/ndkbuild/jni/AIOEither.h, 488
lib/ndkbuild/jni/AIOFifo.c, 492
lib/ndkbuild/jni/AIOFifo.h, 496
lib/ndkbuild/jni/AIOList.c, 500
lib/ndkbuild/jni/AIOList.h, 502
lib/ndkbuild/jni/AIOProductTypes.c, 505
lib/ndkbuild/jni/AIOProductTypes.h, 508
lib/ndkbuild/jni/AIOTuple.c, 510
lib/ndkbuild/jni/AIOTuple.h, 511
lib/ndkbuild/jni/AIOTypes.h, 534
lib/ndkbuild/jni/AIOUSB_ADC.c, 570
lib/ndkbuild/jni/AIOUSB_ADC.h, 598
lib/ndkbuild/jni/AIOUSB_CTR.c, 625
lib/ndkbuild/jni/AIOUSB_CTR.h, 628
lib/ndkbuild/jni/AIOUSB_Core.c, 612
lib/ndkbuild/jni/AIOUSB_Core.h, 619
lib/ndkbuild/jni/AIOUSB_CustomEEPROM.c, 630
lib/ndkbuild/jni/AIOUSB_CustomEEPROM.h, 631
lib/ndkbuild/jni/AIOUSB_DAC.c, 633
lib/ndkbuild/jni/AIOUSB_DAC.h, 636
lib/ndkbuild/jni/AIOUSB_DIO.c, 640
lib/ndkbuild/jni/AIOUSB_DIO.h, 644
lib/ndkbuild/jni/AIOUSB_Log.c, 646
lib/ndkbuild/jni/AIOUSB_Log.h, 649
lib/ndkbuild/jni/AIOUSB_Properties.c, 652
lib/ndkbuild/jni/AIOUSB_Properties.h, 656
lib/ndkbuild/jni/AIOUSB_USB.c, 658
lib/ndkbuild/jni/AIOUSB_USB.h, 658
lib/ndkbuild/jni/AIOUSB_WDG.c, 661
lib/ndkbuild/jni/AIOUSB_WDG.h, 664
lib/ndkbuild/jni/AIOUSBDevice.c, 666
lib/ndkbuild/jni/AIOUSBDevice.h, 669
lib/ndkbuild/jni/CStringArray.c, 680
lib/ndkbuild/jni/CStringArray.h, 681
lib/ndkbuild/jni/DIOBuf.c, 684
lib/ndkbuild/jni/DIOBuf.h, 690
lib/ndkbuild/jni/USBDevice.c, 748
lib/ndkbuild/jni/USBDevice.h, 750
lib/ndkbuild/jni/aiousb.h, 556
lib/ndkbuild/jni/cJSON.c, 672
lib/ndkbuild/jni/cJSON.h, 676
lib/ndkbuild/jni/libaiousb.h, 698
lib/ndkbuild/libusb/README.md, 880
lib/ndkbuild/libusb/Xcode/config.h, 755
lib/ndkbuild/libusb/android/config.h, 754
lib/ndkbuild/libusb/examples/dpfp.c, 756
lib/ndkbuild/libusb/examples/dpfp_threaded.c, 757
lib/ndkbuild/libusb/examples/ezusb.c, 758
lib/ndkbuild/libusb/examples/ezusb.h, 759

lib/ndkbuild/libusb/examples/fxload.c, 761
lib/ndkbuild/libusb/examples/getopt/getopt.c, 761
lib/ndkbuild/libusb/examples/getopt/getopt.h, 762
lib/ndkbuild/libusb/examples/getopt/getopt1.c, 763
lib/ndkbuild/libusb/examples/hotplugtest.c, 764
lib/ndkbuild/libusb/examples/listdevs.c, 764
lib/ndkbuild/libusb/examples/sam3u_benchmark.c, 764
lib/ndkbuild/libusb/examples/testlibusb.c, 765
lib/ndkbuild/libusb/examples/xusb.c, 765
lib/ndkbuild/libusb/libusb/core.c, 767
lib/ndkbuild/libusb/libusb/descriptor.c, 769
lib/ndkbuild/libusb/libusb/hotplug.c, 771
lib/ndkbuild/libusb/libusb/hotplug.h, 771
lib/ndkbuild/libusb/libusb/io.c, 772
lib/ndkbuild/libusb/libusb/libusb.h, 773
lib/ndkbuild/libusb/libusb/libusbi.h, 781
lib/ndkbuild/libusb/libusb/os/darwin_usb.c, 786
lib/ndkbuild/libusb/libusb/os/darwin_usb.h, 786
lib/ndkbuild/libusb/libusb/os/haiku_pollfs.cpp, 787
lib/ndkbuild/libusb/libusb/os/haiku_usb.h, 787
lib/ndkbuild/libusb/libusb/os/haiku_usb_backend.cpp, 788
lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.cpp, 788
lib/ndkbuild/libusb/libusb/os/haiku_usb_raw.h, 788
lib/ndkbuild/libusb/libusb/os/linux_netlink.c, 790
lib/ndkbuild/libusb/libusb/os/linux_udev.c, 790
lib/ndkbuild/libusb/libusb/os/linux_usbfs.c, 791
lib/ndkbuild/libusb/libusb/os/linux_usbfs.h, 792
lib/ndkbuild/libusb/libusb/os/netbsd_usb.c, 795
lib/ndkbuild/libusb/libusb/os/openbsd_usb.c, 795
lib/ndkbuild/libusb/libusb/os/poll_posix.c, 796
lib/ndkbuild/libusb/libusb/os/poll_posix.h, 796
lib/ndkbuild/libusb/libusb/os/poll_windows.c, 797
lib/ndkbuild/libusb/libusb/os/poll_windows.h, 798
lib/ndkbuild/libusb/libusb/os/sunos_usb.c, 801
lib/ndkbuild/libusb/libusb/os/sunos_usb.h, 801
lib/ndkbuild/libusb/libusb/os/threads_posix.c, 802
lib/ndkbuild/libusb/libusb/os/threads_posix.h, 802
lib/ndkbuild/libusb/libusb/os/threads_windows.c, 803
lib/ndkbuild/libusb/libusb/os/threads_windows.h, 804
lib/ndkbuild/libusb/libusb/os/wince_usb.c, 806
lib/ndkbuild/libusb/libusb/os/wince_usb.h, 806
lib/ndkbuild/libusb/libusb/os/windows_common.h, 809
lib/ndkbuild/libusb/libusb/os/windows_nt_common.c, 811
lib/ndkbuild/libusb/libusb/os/windows_nt_common.h, 812
lib/ndkbuild/libusb/libusb/os/windows_usbdk.c, 813
lib/ndkbuild/libusb/libusb/os/windows_usbdk.h, 813
lib/ndkbuild/libusb/libusb/os/windows_winusb.c, 816
lib/ndkbuild/libusb/libusb/os/windows_winusb.h, 818
lib/ndkbuild/libusb/libusb/strerror.c, 831
lib/ndkbuild/libusb/libusb/sync.c, 832
lib/ndkbuild/libusb/libusb/version.h, 832
lib/ndkbuild/libusb/libusb/version_nano.h, 832
lib/ndkbuild/libusb/msvc/config.h, 755
lib/ndkbuild/libusb/msvc/errno.h, 833
lib/ndkbuild/libusb/msvc/inttypes.h, 834
lib/ndkbuild/libusb/msvc/missing.c, 841
lib/ndkbuild/libusb/msvc/missing.h, 841
lib/ndkbuild/libusb/msvc/stdint.h, 841
lib/ndkbuild/libusb/tests/libusb_testlib.h, 845
lib/ndkbuild/libusb/tests/stress.c, 847
lib/ndkbuild/libusb/tests/testlib.c, 847
lib/wrappers/README.doc, 352
lib/wrappers/README.md, 880
lib/wrappers/scilab/foo.c, 848
lib/wrappers/scilab/foo.h, 848
LibK_GetProcAddress_t
windows_winusb.h, 829
LibK_GetVersion_t
windows_winusb.h, 829
libaiousb.h
ADC_ADMode, 735
ADC_BulkAcquire, 736
ADC_BulkPoll, 737
ADC_CanCalibrate, 736
ADC_ClockRateForADCProduct, 736
ADC_CreateFastITConfig, 737
ADC_GetADConfigBlock_Registers, 742
ADC_GetChannelV, 733
ADC_GetConfig, 734
ADC_GetFastITScanV, 737
ADC_GetITScanV, 737
ADC_GetMaxClockRate, 735
ADC_GetOversample, 737
ADC_GetScan, 733
ADC_GetScanV, 732
ADC_InitFastITScanV, 737
ADC_Initialize, 736
ADC_QueryCal, 736
ADC_Range1, 735
ADC_RangeAll, 733
ADC_ResetDevice, 712
ADC_ResetFastITScanV, 737
ADC_SetAllGainCodeAndDiffMode, 740
ADC_SetCal, 736
ADC_SetConfig, 735
ADC_SetFastITScanVChannels, 737
ADC_SetOversample, 738
ADC_SetScanLimits, 735
ADC_VerifyAndCorrectConfigBlock, 721
ADCConfigBlockCopy, 723
ADCConfigBlockGetAIOUSBDevice, 723
ADCConfigBlockGetCalMode, 722
ADCConfigBlockGetClockRate, 722
ADCConfigBlockGetDebug, 723
ADCConfigBlockGetEndChannel, 722
ADCConfigBlockGetGainCode, 721
ADCConfigBlockGetOversample, 722
ADCConfigBlockGetSize, 723
ADCConfigBlockGetStartChannel, 722
ADCConfigBlockGetTesting, 723
ADCConfigBlockGetTimeout, 722
ADCConfigBlockGetTriggerMode, 722
ADCConfigBlockInit, 721
ADCConfigBlockInitForCounterScan, 721
ADCConfigBlockInitializeDefault, 721
ADCConfigBlockInitializeFromAIOUSBDevice, 723
ADCConfigBlockSetAIOUSBDevice, 723
ADCConfigBlockSetAllGainCodeAndDiffMode, 721
ADCConfigBlockSetCalMode, 722
ADCConfigBlockSetChannelRange, 722
ADCConfigBlockSetClockRate, 721
ADCConfigBlockSetDebug, 723
ADCConfigBlockSetDevice, 723
ADCConfigBlockSetDifferentialMode, 723
ADCConfigBlockSetEndChannel, 722
ADCConfigBlockSetGainCode, 721
ADCConfigBlockSetOversample, 722
ADCConfigBlockSetRangeSingle, 723
ADCConfigBlockSetReference, 722
ADCConfigBlockSetRegister, 721
ADCConfigBlockSetScanRange, 722
ADCConfigBlockSetSize, 723
ADCConfigBlockSetStartChannel, 722
ADCConfigBlockSetTesting, 723
ADCConfigBlockSetTimeout, 722
ADCConfigBlockSetTriggerEdge, 723
ADCConfigBlockSetTriggerMode, 722
ADCConfigBlockToJSON, 723
ADCConfigBlockToYAML, 722
AIO_CMDLINE_OPTIONS, 747
AIOBufGetIterator, 725
AIOBufGetRaw, 725
AIOBufGetSize, 725
AIOBufGetType, 725

AIOBufIteratorGetValue, 726
AIOBufIteratorIsValid, 726
AIOBufIteratorNext, 726
AIOBufRead, 725
AIOBufWrite, 725
AIOChannelMaskGetMask, 724
AIOChannelMaskGetMaskAtIndex, 724
AIOChannelMaskGetSize, 725
AIOChannelMaskIndices, 725
AIOChannelMaskNextIndex, 725
AIOChannelMaskNumberChannels, 724
AIOChannelMaskNumberSignals, 725
AIOChannelMaskSetMaskAtIndex, 725
AIOChannelMaskSetMaskFromInt, 725
AIOChannelMaskSetMaskFromStr, 725
AIOChannelMaskToString, 724
AIOChannelMaskToStringAtIndex, 724
AIOChannelRange, 708
AIOChannelRangeGetEnd, 725
AIOChannelRangeGetGain, 725
AIOChannelRangeGetStart, 725
AIOChannelRangeToStr, 725
AIOCommandLineOptionsListDevices, 747
AIOCommandLineOptionsOverrideADCConfigBlock, 747
AIOContinuousBufCallbackStart, 718
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction, 718
AIOContinuousBufCleanup, 720
AIOContinuousBufCountScansAvailable, 720
AIOContinuousBufEnd, 720
AIOContinuousBufGetADCConfigBlock, 716
AIOContinuousBufGetBaseSize, 717
AIOContinuousBufGetBufferSize, 717
AIOContinuousBufGetCallback, 716
AIOContinuousBufGetClock, 720
AIOContinuousBufGetDebug, 717
AIOContinuousBufGetDeviceIndex, 717
AIOContinuousBufGetExitCode, 719
AIOContinuousBufGetNumberChannels, 716
AIOContinuousBufGetNumberSamplesPerScan, 718
AIOContinuousBufGetNumberScans, 717
AIOContinuousBufGetOversample, 717
AIOContinuousBufGetReadPosition, 719
AIOContinuousBufGetRemainingSize, 719
AIOContinuousBufGetRunStatus, 719
AIOContinuousBufGetScansRead, 719
AIOContinuousBufGetStatus, 719
AIOContinuousBufGetStreamingBlockSize, 716
AIOContinuousBufGetTesting, 717
AIOContinuousBufGetTimeout, 717
AIOContinuousBufGetTotalSamplesExpected, 718
AIOContinuousBufGetUnitSize, 717
AIOContinuousBufGetWritePosition, 719
AIOContinuousBufInitADCConfigBlock, 716
AIOContinuousBufInitConfiguration, 716
AIOContinuousBufInitiateCallbackAcquisition, 719
AIOContinuousBufLock, 718
AIOContinuousBufNumberChannels, 717
AIOContinuousBufNumberSamplesAvailable, 718
AIOContinuousBufNumberSignals, 717
AIOContinuousBufNumberWriteSamplesRemaining, 717
AIOContinuousBufPending, 719
AIOContinuousBufPopN, 718
AIOContinuousBufPushN, 718
AIOContinuousBufRead, 720
AIOContinuousBufReadCompleteScanCounts, 719
AIOContinuousBufReadIntegerNumberOfScans, 719
AIOContinuousBufReadIntegerScanCounts, 719
AIOContinuousBufReset, 718
AIOContinuousBufResetDevice, 717
AIOContinuousBufSaveConfig, 717
AIOContinuousBufSendPreConfig, 717
AIOContinuousBufSetAllGainCodeAndDiffMode, 717
AIOContinuousBufSetBaseSize, 717
AIOContinuousBufSetCallback, 716
AIOContinuousBufSetChannelMask, 717
AIOContinuousBufSetChannelRange, 717
AIOContinuousBufSetClock, 720
AIOContinuousBufSetCountsBuffer, 719
AIOContinuousBufSetDebug, 717
AIOContinuousBufSetDeviceIndex, 717
AIOContinuousBufSetDiscardFirstSample, 717
AIOContinuousBufSetNumberChannels, 716
AIOContinuousBufSetNumberScans, 717
AIOContinuousBufSetOversample, 717
AIOContinuousBufSetStartAndEndChannel, 717
AIOContinuousBufSetStreamingBlockSize, 716
AIOContinuousBufSetTesting, 717
AIOContinuousBufSetTimeout, 717
AIOContinuousBufSetUnitSize, 717
AIOContinuousBufSetVoltsBuffer, 720
AIOContinuousBufSimpleSetupConfig, 720
AIOContinuousBufStopAcquisition, 719
AIOContinuousBufToJSON, 720
AIOContinuousBufUnlock, 718
AIOContinuousBufWrite, 720
AIOContinuousBufWriteCounts, 720
AIOCountsConverter, 708
AIOCountsConverterConvert, 726
AIOCountsConverterConvertAllAvailableScans, 726
AIOCountsConverterConvertFifo, 726
AIOCountsConverterConvertNScans, 726
AIOCountsConverterReset, 726
AIODeviceInfoGet, 727
AIODeviceInfoGetCounters, 727
AIODeviceInfoGetDIOBytes, 727
AIODeviceInfoGetName, 727
AIODeviceQueryGetIndex, 728
AIODeviceQueryGetName, 728
AIODeviceQueryGetNumCounters, 728
AIODeviceQueryGetNumDIOBytes, 728
AIODeviceQueryGetProductID, 727
AIODeviceQueryNameSize, 728
AIODeviceQueryToRepr, 727
AIODeviceQueryToStr, 727
AIODeviceTableAddDeviceToDeviceTable, 728
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice, 729
AIODeviceTableClearDevices, 729
AIODeviceTableGetAIOUSBDeviceAtIndex, 729
AIODeviceTableGetDeviceAtIndex, 729
AIODeviceTableGetUSBDeviceAtIndex, 729
AIODeviceTableInit, 731
AIODeviceTablePopulateTable, 729
AIODeviceTablePopulateTableTest, 729
AIOEitherClear, 732
AIOEitherGetLeft, 732
AIOEitherGetRight, 732
AIOEitherHasError, 732
AIOEitherSetLeft, 732
AIOEitherSetRight, 732
AIOEitherToAIONumber, 732
AIOEitherToAIORetType, 732
AIOEitherToDouble, 732
AIOEitherToInt, 732
AIOEitherToShort, 732
AIOEitherToString, 732
AIOEitherToUnsigned, 732
AIOFifoGetRefSize, 731
AIOFifoGetSize, 732
AIOFifoGetSizeNumElements, 732
AIOFifoRead, 731

AIOFifoReadAllOrNone, 731
AIOFifoReadPosition, 732
AIOFifoReadSize, 732
AIOFifoReadSizeNumElements, 732
AIOFifoReset, 731
AIOFifoResize, 732
AIOFifoWrite, 731
AIOFifoWriteAllOrNone, 731
AIOFifoWritePosition, 732
AIOFifoWriteSizeRemaining, 732
AIOFifoWriteSizeRemainingNumElements, 732
AIOGetChannelRange, 747
AIOPrintUsage, 747
AIOProcessCmdline, 746
AIOProcessCommandLine, 746
AIORESULT, 708
AIORET_TYPE, 708
AIOUSB_ADC_ExternalCal, 738
AIOUSB_ADC_InternalCal, 741
AIOUSB_ADC_LoadCalTable, 715
AIOUSB_ADC_SetCalTable, 715
AIOUSB_ClearFIFO, 715
AIOUSB_EnsureOpen, 731
AIOUSB_Exit, 731
AIOUSB_FindDeviceIndicesByGroup, 745
AIOUSB_FindDevices, 744
AIOUSB_FindDevicesByGroup, 744
AIOUSB_GetAllDevices, 731
AIOUSB_GetCalMode, 740
AIOUSB_GetCommTimeout, 713
AIOUSB_GetConfigBlock, 713
AIOUSB_GetDevice, 713
AIOUSB_GetDeviceByProductID, 744
AIOUSB_GetDeviceProperties, 744
AIOUSB_GetDeviceSerialNumber, 713
AIOUSB_GetEndChannel, 741
AIOUSB_GetGainCode, 738
AIOUSB_GetMiscClock, 713
AIOUSB_GetOversample, 740
AIOUSB_GetRegister, 742
AIOUSB_GetResultCodeAsString, 713
AIOUSB_GetScan, 734
AIOUSB_GetStartChannel, 741
AIOUSB_GetStreamingBlockSize, 715
AIOUSB_GetTriggerMode, 740
AIOUSB_GetVersion, 713
AIOUSB_GetVersionDate, 713
AIOUSB_INIT_PATTERN, 748
AIOUSB_Init, 730
AIOUSB_InitConfigBlock, 715
AIOUSB_InitTest, 713
AIOUSB_IsDifferentialMode, 738
AIOUSB_IsDiscardFirstSample, 741
AIOUSB_IsInit, 731
AIOUSB_ListDevices, 744
AIOUSB_Lock, 713
AIOUSB_Reset, 731
AIOUSB_ResetChip, 713
AIOUSB_SetAllGainCodeAndDiffMode, 738
AIOUSB_SetCalMode, 740
AIOUSB_SetCommTimeout, 713
AIOUSB_SetDifferentialMode, 740
AIOUSB_SetDiscardFirstSample, 741
AIOUSB_SetGainCode, 738
AIOUSB_SetMiscClock, 713
AIOUSB_SetOversample, 740
AIOUSB_SetRegister, 742
AIOUSB_SetScanRange, 741
AIOUSB_SetStreamingBlockSize, 741
AIOUSB_SetTriggerMode, 740
AIOUSB_ShowDevices, 744
AIOUSB_UnLock, 713
AIOUSB_Validate, 713
AIOUSB_Validate_Device, 715
AIOUSB_Validate_Lock, 713
AIOUSB_VoltsToCounts, 713
AIOUSBDeviceCopyADCConfigBlock, 743
AIOUSBDeviceGetADCConfigBlock, 743
AIOUSBDeviceGetDiscardFirstSample, 743
AIOUSBDeviceGetStreamingBlockSize, 743
AIOUSBDeviceGetTesting, 743
AIOUSBDeviceGetTimeout, 744
AIOUSBDeviceGetUSBHandle, 743
AIOUSBDeviceGetUSBHandleFromDeviceIndex, 743
AIOUSBDeviceInitializeWithProductID, 743
AIOUSBDeviceSetADCConfigBlock, 743
AIOUSBDeviceSetDiscardFirstSample, 744
AIOUSBDeviceSetTesting, 743
AIOUSBDeviceSetTimeout, 744
AIOUSBDeviceSetUSBHandle, 743
AIOUSBDeviceSize, 743
AIOUSBDeviceToJSON, 743
AIOUSBDeviceWriteADCConfig, 744
AIOUSBGetError, 731
AddAllACCESUSBDevices, 745
aio_channel_range_error, 748
CTR_8254Load, 721
CTR_8254Mode, 721
CTR_8254ModeLoad, 721
CTR_8254Read, 721
CTR_8254ReadAll, 721
CTR_8254ReadLatched, 721
CTR_8254ReadModeLoad, 721
CTR_8254ReadStatus, 721
CTR_8254SelectGate, 721
CTR_CalculateCountersForClock, 720
CTR_StartOutputFreq, 721
ClearAIODeviceTable, 731
ClearDevices, 729
CloseAllDevices, 731
CopyUSBDevice, 745
CreateSmartBuffer, 741
CustomEEPROMRead, 743
CustomEEPROMWrite, 743
DACDirect, 742
DACMultiDirect, 742
DACOutputClose, 743
DACOutputCloseNoEnd, 743
DACOutputFrame, 743
DACOutputFrameRaw, 743
DACOutputOpen, 743
DACOutputSetCount, 743
DACOutputSetInterlock, 743
DACOutputStart, 743
DACSetBoardRange, 743
DIO_ConfigurationQuery, 711
DIO_Configure, 711
DIO_ConfigureEx, 711
DIO_ConfigureWithDIOBuf, 711
DIO_Read1, 712
DIO_Read8, 712
DIO_ReadAll, 712
DIO_ReadAllToCharStr, 712
DIO_ReadAllToDIOBuf, 712
DIO_ReadIntoDIOBuf, 712
DIO_StreamClose, 712
DIO_StreamFrame, 712
DIO_StreamOpen, 712
DIO_StreamSetClocks, 712
DIO_Write1, 712
DIO_Write8, 711
DIO_WriteAll, 711
DIOBufByteSize, 711

DIOBufGetByteAtIndex, 711
DIOBufGetIndex, 711
DIOBufReplaceBinString, 709
DIOBufReplaceString, 709
DIOBufResize, 711
DIOBufSetByteAtIndex, 711
DIOBufSetIndex, 711
DIOBufSize, 711
DIOBufToBinary, 709
DIOBufToHex, 709
DIOBufToInvertedBinary, 710
DIOBufToString, 711
DeleteADCConfigBlock, 723
DeleteAIOBuf, 725
DeleteAIOChannelMask, 723
DeleteAIOChannelRange, 725
DeleteAIOCommandLineOptions, 747
DeleteAIOContinuousBuf, 716
DeleteAIOCountsConverter, 726
DeleteAIODeviceInfo, 727
DeleteAIODeviceQuery, 727
DeleteAIOFifo, 731
DeleteAIOGainRange, 727
DeleteDIOBuf, 709
DeleteUSBDevice, 745
DeleteUSBDevices, 746
DeleteWDGConfig, 745
DeviceTableAtIndex, 713
DeviceTableAtIndex_Lock, 713
GenericVendorRead, 715
GenericVendorWrite, 715
get_usb_device, 746
GetDeviceBySerialNumber, 744
GetDeviceSerialNumber, 744
GetDevices, 729
GetSafeDeviceName, 730
InitializeUSBDevice, 745
is_all_digits, 723
MATLAB, 708
NewADCConfigBlockFromJSON, 723
NewAIOBuf, 725
NewAIOChannelMask, 723
NewAIOChannelMaskFromChr, 724
NewAIOChannelMaskFromStr, 723
NewAIOChannelRangeFromStr, 725
NewAIOCommandLineOptionsFromDefaultOptions, 746
NewAIOContinuousBuf, 716
NewAIOContinuousBufForCounts, 716
NewAIOContinuousBufForVolts, 716
NewAIOContinuousBufFromJSON, 720
NewAIOCountsConverter, 726
NewAIOCountsConverterFromAIOContinuousBuf, 726
NewAIOCountsConverterWithBuffer, 726
NewAIOCountsConverterWithScanLimiter, 726
NewAIODeviceInfo, 727
NewAIODeviceQuery, 727
NewAIOFifo, 731
NewAIOFifoTYPE, 731
NewAIOGainRangeFromADCConfigBlock, 726
NewAIOUSBDeviceFromJSON, 743
NewDIOBuf, 708
NewDIOBufFromBinStr, 709
NewDIOBufFromChar, 709
NewDefaultAIOCommandLineOptions, 746
NewUSBDevice, 745
NewWDGConfig, 745
PopN, 731
ProductIDToName, 730
ProductNameToID, 730
Push, 731
PushN, 731
QueryDeviceInfo, 729
ReadConfigBlock, 738
USBDeviceClose, 746
USBDeviceFetchADCConfigBlock, 746
USBDeviceGetIdProduct, 746
USBDeviceGetUSBDeviceHandle, 746
USBDevicePutADCConfigBlock, 746
usb_bulk_transfer, 746
usb_control_transfer, 746
usb_request, 746
usb_reset_device, 746
WDG_GetStatus, 745
WDG_Pet, 745
WDG_SetConfig, 745
WriteConfigBlock, 738
Library initialization/deinitialization, 89
LIBUSB_LOG_LEVEL_DEBUG, 90
LIBUSB_LOG_LEVEL_ERROR, 90
LIBUSB_LOG_LEVEL_INFO, 90
LIBUSB_LOG_LEVEL_NONE, 90
LIBUSB_LOG_LEVEL_WARNING, 90
libusb_context, 89
libusb_exit, 90
libusb_init, 90
libusb_log_level, 90
libusb_set_debug, 90
libusb, 29
libusb.h
LIBUSB_DT_BOS_SIZE, 780
LIBUSB_ERROR_COUNT, 780
LIBUSBX_API_VERSION, 780
libusb_testlib.h
TEST_STATUS_ERROR, 846
TEST_STATUS_FAILURE, 846
TEST_STATUS_SKIP, 846
TEST_STATUS_SUCCESS, 846
libusb_alloc_streams
Asynchronous device I/O, 126
libusb_alloc_transfer
Asynchronous device I/O, 127
libusb_attach_kernel_driver
Device handling and enumeration, 104
libusb_bos_descriptor, 235
bDescriptorType, 235
bLength, 235
bNumDeviceCaps, 235
dev_capability, 235
wTotalLength, 235
libusb_bos_dev_capability_descriptor, 236
bDescriptorType, 236
bDevCapabilityType, 236
bLength, 236
dev_capability_data, 236
libusb_bos_type
Device handling and enumeration, 95
libusb_bulk_transfer
Synchronous device I/O, 140
libusb_cancel_transfer
Asynchronous device I/O, 128
libusb_capability
Miscellaneous, 107
libusb_claim_interface
Device handling and enumeration, 100
libusb_class_code
USB descriptors, 111
libusb_clear_halt
Device handling and enumeration, 101
libusb_close
Device handling and enumeration, 99
libusb_config_descriptor, 236
bConfigurationValue, 237

bDescriptorType, 237
bLength, 237
bNumInterfaces, 237
bmAttributes, 237
extra, 238
extra_length, 238
iConfiguration, 237
interface, 238
MaxPower, 237
wTotalLength, 237
libusb_container_id_descriptor, 238
bDescriptorType, 238
bDevCapabilityType, 238
bLength, 238
bReserved, 239
ContainerID, 239
libusb_context, 239
completed_transfers, 240
debug, 239
debug_fixed, 239
device_close, 240
event_data_lock, 240
event_flags, 240
event_handler_active, 240
event_handling_key, 240
event_pipe, 239
event_waiters_cond, 240
event_waiters_lock, 240
events_lock, 240
fd_added_cb, 240
fd_cb_user_data, 240
fd_removed_cb, 240
flying_transfers, 240
flying_transfers_lock, 240
hotplug_cbs, 240
hotplug_cbs_lock, 240
hotplug_msgs, 240
ipollfds, 240
Library initialization/deinitialization, 89
list, 240
open_devs, 239
open_devs_lock, 240
pollfds, 240
pollfds_cnt, 240
usb_devs, 239
usb_devs_lock, 239
libusb_control_setup, 240
bRequest, 241
bmRequestType, 241
wIndex, 241
wLength, 241
wValue, 241
libusb_control_transfer
Synchronous device I/O, 140
libusb_descriptor_type
USB descriptors, 112
libusb_detach_kernel_driver
Device handling and enumeration, 103
libusb_dev_mem_alloc
Asynchronous device I/O, 127
libusb_dev_mem_free
Asynchronous device I/O, 127
libusb_device, 241
attached, 242
bus_number, 242
ctx, 242
Device handling and enumeration, 94
device_address, 242
device_descriptor, 242
list, 242
lock, 242
num_configurations, 242
os_priv, 242
parent_dev, 242
port_number, 242
refcnt, 242
session_data, 242
speed, 242
libusb_device_descriptor, 242
bDescriptorType, 243
bDeviceClass, 243
bDeviceProtocol, 243
bDeviceSubClass, 243
bLength, 243
bMaxPacketSize0, 243
bNumConfigurations, 244
bcdDevice, 243
bcdUSB, 243
iManufacturer, 243
iProduct, 243
iSerialNumber, 244
idProduct, 243
idVendor, 243
libusb_device_handle, 244
auto_detach_kernel_driver, 244
claimed_interfaces, 244
dev, 244
Device handling and enumeration, 94
list, 244
lock, 244
os_priv, 244
libusb_endpoint_descriptor, 244
bDescriptorType, 245
bEndpointAddress, 245
bInterval, 245
bLength, 245
bRefresh, 245
bSynchAddress, 245
bmAttributes, 245
extra, 245
extra_length, 246
wMaxPacketSize, 245
libusb_endpoint_direction
USB descriptors, 112
libusb_error
Miscellaneous, 107
libusb_error_name
Miscellaneous, 108
libusb_event_handler_active
Polling and timing, 135
libusb_event_handling_ok
Polling and timing, 135
libusb_exit
Library initialization/deinitialization, 90
libusb_free_bos_descriptor
USB descriptors, 115
libusb_free_config_descriptor
USB descriptors, 114
libusb_free_container_id_descriptor
USB descriptors, 117
libusb_free_device_list
Device handling and enumeration, 96
libusb_free_pollfds
Polling and timing, 139
libusb_free_ss_endpoint_companion_descriptor
USB descriptors, 114
libusb_free_ss_usb_device_capability_descriptor
USB descriptors, 117
libusb_free_streams
Asynchronous device I/O, 127
libusb_free_transfer
Asynchronous device I/O, 128
libusb_free_usb_2_0_extension_descriptor
USB descriptors, 115

libusb_get_active_config_descriptor
USB descriptors, 113
libusb_get_bos_descriptor
USB descriptors, 115
libusb_get_bus_number
Device handling and enumeration, 96
libusb_get_config_descriptor
USB descriptors, 113
libusb_get_config_descriptor_by_value
USB descriptors, 114
libusb_get_configuration
Device handling and enumeration, 99
libusb_get_container_id_descriptor
USB descriptors, 117
libusb_get_device
Device handling and enumeration, 99
libusb_get_device_address
Device handling and enumeration, 97
libusb_get_device_descriptor
USB descriptors, 113
libusb_get_device_list
Device handling and enumeration, 96
libusb_get_device_speed
Device handling and enumeration, 97
libusb_get_max_iso_packet_size
Device handling and enumeration, 98
libusb_get_max_packet_size
Device handling and enumeration, 97
libusb_get_next_timeout
Polling and timing, 138
libusb_get_parent
Device handling and enumeration, 97
libusb_get_pollfds
Polling and timing, 139
libusb_get_port_number
Device handling and enumeration, 96
libusb_get_port_numbers
Device handling and enumeration, 96
libusb_get_port_path
Device handling and enumeration, 97
libusb_get_ss_endpoint_companion_descriptor
USB descriptors, 114
libusb_get_ss_usb_device_capability_descriptor
USB descriptors, 115
libusb_get_string_descriptor_ascii
USB descriptors, 117
libusb_get_usb_2_0_extension_descriptor
USB descriptors, 115
libusb_get_version
Miscellaneous, 108
libusb_handle_events
Polling and timing, 137
libusb_handle_events_completed
Polling and timing, 137
libusb_handle_events_locked
Polling and timing, 137
libusb_handle_events_timeout
Polling and timing, 136
libusb_handle_events_timeout_completed
Polling and timing, 136
libusb_has_capability
Miscellaneous, 108
libusb_hid_descriptor, 246
bClassDescriptorType, 246
bCountryCode, 246
bDescriptorType, 246
bLength, 246
bNumDescriptors, 246
bcdHID, 246
wClassDescriptorLength, 246
libusb_hid_report_type
windows_winusb.h, 829
libusb_hid_request_type
windows_winusb.h, 829
libusb_hotplug_callback, 246
cb, 247
ctx, 247
dev_class, 247
events, 247
flags, 247
handle, 247
hotplug.h, 772
list, 247
needs_free, 247
product_id, 247
user_data, 247
vendor_id, 247
libusb_hotplug_callback_fn
Device hotplug event notification, 118
libusb_hotplug_callback_handle
Device hotplug event notification, 118
libusb_hotplug_deregister_callback
Device hotplug event notification, 120
libusb_hotplug_event
Device hotplug event notification, 119
libusb_hotplug_flag
Device hotplug event notification, 119
libusb_hotplug_message, 248
device, 248
event, 248
hotplug.h, 772
list, 248
libusb_hotplug_register_callback
Device hotplug event notification, 119
libusb_init
Library initialization/deinitialization, 90
libusb_interface, 248
altsetting, 248
num_altsetting, 248
libusb_interface_descriptor, 249
bAlternateSetting, 249
bDescriptorType, 249
bInterfaceClass, 250
bInterfaceNumber, 249
bInterfaceProtocol, 250
bInterfaceSubClass, 250
bLength, 249
bNumEndpoints, 250
endpoint, 250
extra, 250
extra_length, 250
iInterface, 250
libusb_interrupt_event_handler
Polling and timing, 135
libusb_interrupt_transfer
Synchronous device I/O, 141
libusb_iso_packet_descriptor, 250
actual_length, 251
length, 251
status, 251
libusb_iso_sync_type
USB descriptors, 112
libusb_iso_usage_type
USB descriptors, 113
libusb_kernel_driver_active
Device handling and enumeration, 103
libusb_le16_to_cpu
Miscellaneous, 106
libusb_lock_event_waiters
Polling and timing, 135
libusb_lock_events
Polling and timing, 134
libusb_log_level
Library initialization/deinitialization, 90

libusb_open
Device handling and enumeration, 98
libusb_open_device_with_vid_pid
Device handling and enumeration, 99
libusb_pollfd, 251
events, 251
fd, 251
libusb_pollfd_added_cb
Polling and timing, 133
libusb_pollfd_removed_cb
Polling and timing, 134
libusb_pollfds_handle_timeouts
Polling and timing, 137
libusb_ref_device
Device handling and enumeration, 98
libusb_release_interface
Device handling and enumeration, 101
libusb_request_recipient
Miscellaneous, 107
libusb_request_type
Miscellaneous, 107
libusb_reset_device
Device handling and enumeration, 103
libusb_set_auto_detach_kernel_driver
Device handling and enumeration, 104
libusb_set_configuration
Device handling and enumeration, 100
libusb_set_debug
Library initialization/deinitialization, 90
libusb_set_interface_alt_setting
Device handling and enumeration, 101
libusb_set_pollfd_notifiers
Polling and timing, 138
libusb_setlocale
Miscellaneous, 108
libusb_speed
Device handling and enumeration, 95
libusb_ss_endpoint_companion_descriptor, 251
bDescriptorType, 252
bLength, 252
bMaxBurst, 252
bmAttributes, 252
wBytesPerInterval, 252
libusb_ss_usb_device_capability_attributes
Device handling and enumeration, 95
libusb_ss_usb_device_capability_descriptor, 252
bDescriptorType, 253
bDevCapabilityType, 253
bFunctionalitySupport, 253
bLength, 253
bU1DevExitLat, 253
bU2DevExitLat, 253
bmAttributes, 253
wSpeedSupported, 253
libusb_standard_request
Miscellaneous, 106
libusb_strerror
Miscellaneous, 108
libusb_submit_transfer
Asynchronous device I/O, 128
libusb_supported_speed
Device handling and enumeration, 95
libusb_testlib.h
bool, 846
false, 846
LIBUSB_NULL_TEST, 846
libusb_testlib_logf, 847
libusb_testlib_result, 846
libusb_testlib_run_tests, 847
libusb_testlib_test_function, 846
true, 846
libusb_testlib_ctx, 254
list_tests, 254
null_fd, 254
old_stderr, 254
old_stdout, 254
output_file, 254
test_count, 254
test_names, 254
verbose, 254
libusb_testlib_logf
libusb_testlib.h, 847
testlib.c, 848
libusb_testlib_result
libusb_testlib.h, 846
libusb_testlib_run_tests
libusb_testlib.h, 847
testlib.c, 848
libusb_testlib_test, 254
function, 254
name, 254
libusb_testlib_test_function
libusb_testlib.h, 846
libusb_transfer, 255
actual_length, 256
buffer, 256
callback, 256
dev_handle, 255
endpoint, 255
flags, 255
iso_packet_desc, 256
length, 256
num_iso_packets, 256
status, 256
timeout, 256
type, 255
user_data, 256
libusb_transfer_flags
Asynchronous device I/O, 126
libusb_transfer_get_stream_id
Asynchronous device I/O, 130
libusb_transfer_set_stream_id
Asynchronous device I/O, 128
libusb_transfer_status
Asynchronous device I/O, 126
libusb_transfer_type
USB descriptors, 112
libusb_try_lock_events
Polling and timing, 134
libusb_unlock_event_waiters
Polling and timing, 136
libusb_unlock_events
Polling and timing, 134
libusb_unref_device
Device handling and enumeration, 98
libusb_usb_2_0_extension_attributes
Device handling and enumeration, 95
libusb_usb_2_0_extension_descriptor, 256
bDescriptorType, 257
bDevCapabilityType, 257
bLength, 257
bmAttributes, 257
libusb_version, 257
describe, 258
major, 258
micro, 258
minor, 258
nano, 258
rc, 258
libusb_wait_for_event
Polling and timing, 136
libusbi.h
USBI_CLOCK_MONOTONIC, 784
USBI_CLOCK_REALTIME, 784

USBI_EVENT_POLLFDS_MODIFIED, 784
USBI_EVENT_USER_INTERRUPT, 784
USBI_TRANSFER_CANCELLING, 784
USBI_TRANSFER_DEVICE_DISAPPEARED, 784
USBI_TRANSFER_IN_FLIGHT, 784
USBI_TRANSFER_OS_HANDLES_TIMEOUT, 785
USBI_TRANSFER_TIMED_OUT, 785
USBI_TRANSFER_TIMEOUT_HANDLED, 785
libusbi.h
API_EXPORTED, 782
ARRAYSIZE, 783
active_contexts_list, 786
active_contexts_lock, 786
container_of, 783
DEVICE_CTX, 783
DEVICE_DESC_LENGTH, 782
darwin_backend, 785
discovered_devs_append, 785
HANDLE_CTX, 784
haiku_usb_raw_backend, 786
IS_EPIN, 784
IS_EPOUT, 784
IS_XFERIN, 784
IS_XFEROUT, 784
ITRANSFER_CTX, 784
linux_usbfs_backend, 785
list_empty, 783
list_entry, 783
list_first_entry, 783
list_for_each_entry, 783
list_for_each_entry_safe, 783
MAX, 783
MIN, 783
netbsd_backend, 785
openbsd_backend, 785
sunos_backend, 786
TIMESPEC_IS_SET, 783
TIMESPEC_TO_TIMEVAL, 783
TRANSFER_CTX, 784
UNUSED, 783
USB_MAXCONFIG, 782
USB_MAXENDPOINTS, 782
USB_MAXINTERFACES, 782
USBI_GET_CONTEXT, 783
USBI_LOG_LINE_END, 783
USBI_MAX_LOG_LEN, 782
usbdk_backend, 785
usbi_add_pollfd, 785
usbi_alloc_device, 785
usbi_backend, 785
usbi_clear_event, 785
usbi_connect_device, 785
usbi_dbg, 783
usbi_default_context, 785
usbi_device_cache_descriptor, 785
usbi_disconnect_device, 785
usbi_end_event_handling, 784
usbi_err, 783
usbi_event_flags, 784
usbi_get_config_index_by_value, 785
usbi_get_device_by_session_id, 785
usbi_handle_disconnect, 785
usbi_handle_transfer_cancellation, 785
usbi_handle_transfer_completion, 785
usbi_handling_events, 784
usbi_info, 783
usbi_io_exit, 785
usbi_io_init, 785
usbi_log, 785
usbi_log_v, 785
usbi_parse_descriptor, 785
usbi_pending_events, 784
usbi_remove_pollfd, 785
usbi_sanitize_device, 785
usbi_signal_event, 785
usbi_signal_transfer_completion, 785
usbi_start_event_handling, 784
usbi_transfer_state_flags, 784
usbi_transfer_timeout_flags, 784
usbi_using_timerfd, 784
usbi_warn, 783
wince_backend, 786
windows_backend, 785
linux_usbfs.c
CANCELLED, 792
COMPLETED_EARLY, 792
ERROR, 792
NORMAL, 791
SUBMIT_FAILED, 791
linux_usbfs.h
USBFS_URB_TYPE_BULK, 794
USBFS_URB_TYPE_CONTROL, 794
USBFS_URB_TYPE_INTERRUPT, 794
USBFS_URB_TYPE_ISO, 794
linux_device_disconnected
linux_usbfs.c, 792
linux_usbfs.h, 795
linux_device_handle_priv, 258
caps, 258
fd, 258
fd_removed, 258
linux_device_priv, 259
active_config, 259
descriptors, 259
descriptors_len, 259
sysfs_dir, 259
linux_enumerate_device
linux_usbfs.c, 792
linux_usbfs.h, 795
linux_get_device_address
linux_usbfs.c, 792
linux_usbfs.h, 795
linux_hotplug_enumerate
linux_usbfs.c, 792
linux_usbfs.h, 795
linux_hotplug_lock
linux_usbfs.c, 792
linux_usbfs.h, 795
linux_netlink.c
linux_netlink_hotplug_poll, 790
linux_netlink_start_event_monitor, 790
linux_netlink_stop_event_monitor, 790
NL_GROUP_KERNEL, 790
linux_netlink_hotplug_poll
linux_netlink.c, 790
linux_usbfs.h, 795
linux_netlink_start_event_monitor
linux_netlink.c, 790
linux_usbfs.h, 795
linux_netlink_stop_event_monitor
linux_netlink.c, 790
linux_usbfs.h, 795
linux_transfer_priv, 259
iso_packet_offset, 259
iso_urbs, 259
num_retired, 259
num_urbs, 259
reap_action, 259
reap_status, 259
urbs, 259
linux_udev.c
linux_udev_hotplug_poll, 791
linux_udev_scan_devices, 791
linux_udev_start_event_monitor, 791

linux_udev_stop_event_monitor, 791
linux_udev_hotplug_poll
linux_udev.c, 791
linux_udev_scan_devices
linux_udev.c, 791
linux_udev_start_event_monitor
linux_udev.c, 791
linux_udev_stop_event_monitor
linux_udev.c, 791
linux_usbfs.c
linux_device_disconnected, 792
linux_enumerate_device, 792
linux_get_device_address, 792
linux_hotplug_enumerate, 792
linux_hotplug_lock, 792
linux_usbfs_backend, 792
reap_action, 791
linux_usbfs.h
IOCTL_USBFS_BULK, 794
IOCTL_USBFS_IOCTL, 794
IOCTL_USBFS_RESET, 794
linux_device_disconnected, 795
linux_enumerate_device, 795
linux_get_device_address, 795
linux_hotplug_enumerate, 795
linux_hotplug_lock, 795
linux_netlink_hotplug_poll, 795
linux_netlink_start_event_monitor, 795
linux_netlink_stop_event_monitor, 795
SYSFS_DEVICE_PATH, 793
usbfs_urb_type, 794
linux_usbfs_backend
libusbi.h, 785
linux_usbfs.c, 792
list
darwin_cached_device, 213
driver_lookup, 229
libusb_context, 240
libusb_device, 242
libusb_device_handle, 244
libusb_hotplug_callback, 247
libusb_hotplug_message, 248
usbi_cond_perthread, 331
usbi_pollfd, 333
usbi_transfer, 333
list_empty
libusbi.h, 783
list_entry
libusbi.h, 783
list_first_entry
libusbi.h, 783
list_for_each_entry
libusbi.h, 783
list_for_each_entry_safe
libusbi.h, 783
list_head, 259
next, 260
prev, 260
list_tests
libusb_testlib_ctx, 254
listDevices
AIOUSB::USBDeviceManager, 323
listdevs.c
main, 764
loc
aiobuf_iterator, 156
Location
USBDevice, 313
location
darwin_cached_device, 213
lock
AIOContinuousBuf, 160
libusb_device, 242
libusb_device_handle, 244
usbi_transfer, 334
logerror
ezusb.c, 759
fxload.c, 761
lookup, 260
str, 260
strvalue, 260
value, 260
lookup_voltage_range
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
LowSpeed
windows_usbdk.h, 815
LowSpeed
windows_winusb.h, 826

MAX_IMM_ADCS
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
MAX_USB_DEVICES
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
MODE_AWAIT_FINGER_OFF
dpfp.c, 756
dpfp_threaded.c, 758
MODE_AWAIT_FINGER_ON
dpfp.c, 756
dpfp_threaded.c, 758
MODE_CAPTURE
dpfp.c, 756
dpfp_threaded.c, 758
MODE_INIT
dpfp.c, 756
dpfp_threaded.c, 758
MODE_READY
dpfp.c, 756
dpfp_threaded.c, 758
MODE_SHUT_UP
dpfp.c, 756
dpfp_threaded.c, 758
MAGENTA
AIOUSB_Log.h, 647
ndkbuild/jni/AIOUSB_Log.h, 649
MAKE_MASK
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 875
MASK_BYTES
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 876
MASK_BYTES_SIZE
AIOUSB_DIO.c, 638
MATLAB
libaiousb.h, 708
MAX
AIOTypes.h, 518
libusbi.h, 783
ndkbuild/jni/AIOTypes.h, 540
MAX_CHANNELS
AIOUSB::AnalogInputSubsystem, 187
TestCaseSetup, 275
MAX_COUNTS
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
MAX_DEVICE_COUNT
wince_usb.h, 807

MAX_DIO_BYTES
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 876
MAX_ENUM_GUIDS
windows_winusb.c, 817
MAX_FDS
poll_windows.h, 799
MAX_KEY_LENGTH
windows_winusb.h, 823
MAX_NAME_SIZE
dio96_read_write.c, 871
mytest.c, 873
TestCaseSetup.h, 852
tmp.c, 874
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 876
MAX_OS_FD_LENGTH
xusb.c, 767
MAX_OVERSAMPLE
AIOUSB::AnalogInputSubsystem, 187
MAX_PATH_LENGTH
windows_winusb.h, 823
MAX_PRODUCT_ID
AIOUSB::USBDeviceManager, 325
MAX_USB_DEVICES
windows_winusb.h, 823
MAX_VALUE
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
METHOD_BUFFERED
windows_winusb.h, 825
MIN
AIOTypes.h, 518
libusbi.h, 783
ndkbuild/jni/AIOTypes.h, 540
MIN_COUNTS
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
MIN_PRODUCT_ID
AIOUSB::USBDeviceManager, 325
MIN_VALUE
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
MODE_ONE_SHOT
AIOUSB::Counter, 203
MODE_SQUARE_WAVE
AIOUSB::Counter, 203
main
bulk_acquire_sample.c, 853
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 856
daisample.c, 867
daitest.cpp, 856
dio96_read_write.c, 872
dio_sample.c, 856
diotest.c, 857
diotest2.cpp, 857
dirktest.c, 858
dpfp.c, 757
dpfp_threaded.c, 758
extcal.cpp, 861
fxload.c, 761
HOLD/reverse_cal_table.cpp, 858
hotplugtest.c, 764
idio_sample.c, 878
idio_sample2.c, 879
iiro_sample.c, 880
jni/read_channels_test.c, 861
julian_test.c, 858
listdevs.c, 764
mytest.c, 874
perftest.c, 879
read_and_write.c, 871
read_channels_test.c, 862
read_channels_with_getchannelv_test.cpp, 862
receiver.cpp, 867
reverse_cal_table.cpp, 859
sam3u_benchmark.c, 765
sample3.c, 870
sample_dio.c, 859
SampleClass.cpp, 866
simp_test.cpp, 864
simple_continuous_with_json.c, 865
slow_receiver_test.cpp, 859
standalone_receiver.c, 867
start_stop_continuous.c, 865
stream_test.c, 866
stress.c, 847
test.c, 860
test_fastscan.cpp, 866
testlibusb.c, 765
tmp.c, 875
USB-AI16-16/sample.cpp, 863
USB-AO16-16/sample.cpp, 863
USB-DA12-8A/sample.cpp, 863
USB-DIO-16/sample.cpp, 864
USB-DIO-32/read_and_write_sample.c, 867
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 877
xusb.c, 767
major
libusb_version, 258
malloc_fn
cJSON_Hooks, 198
mask
AIOContinuousBuf, 161
max
AIOGainRange, 163
max_channels
opts, 264
maxCounts
AIOUSB::AnalogIORange, 190
AIOUSB::AnalogOutputSubsystem, 192
MaxPower
libusb_config_descriptor, 237
USB_CONFIGURATION_DESCRIPTOR, 286
maxVolts
AIOUSB::AnalogIORange, 190
maxcount
options, 263
maxcounts
TestCaseSetup, 275
MaximumPacketSize
WINUSB_PIPE_INFORMATION, 342
maxvalue
rangelookup, 267
message
Error, 230
message_lock
AIOUSB_Log.c, 646
AIOUSB_Log.h, 649
ndkbuild/jni/AIOUSB_Log.c, 647
ndkbuild/jni/AIOUSB_Log.h, 651
MessageReceived
RosterLooper, 267
MiParentInformation
USB_NODE_INFORMATION, 307
micro
libusb_version, 258
min

AIOGainRange, 163
minCounts
AIOUSB::AnalogIORange, 190
AIOUSB::AnalogOutputSubsystem, 192
minVolts
ADRange, 146
AIOUSB::AnalogIORange, 190
minor
devlink_cbarg, 219
libusb_version, 258
minvalue
rangelookup, 267
miscClockHz
AIOUSBDevice, 166
Miscellaneous, 105
LIBUSB_CAP_HAS_CAPABILITY, 107
LIBUSB_CAP_HAS_HID_ACCESS, 107
LIBUSB_CAP_HAS_HOTPLUG, 107
LIBUSB_CAP_SUPPORTS_DETACH_KERNEL_DRIVER, 107
LIBUSB_ERROR_ACCESS, 107
LIBUSB_ERROR_BUSY, 107
LIBUSB_ERROR_INTERRUPTED, 107
LIBUSB_ERROR_INVALID_PARAM, 107
LIBUSB_ERROR_IO, 107
LIBUSB_ERROR_NO_DEVICE, 107
LIBUSB_ERROR_NO_MEM, 107
LIBUSB_ERROR_NOT_FOUND, 107
LIBUSB_ERROR_NOT_SUPPORTED, 107
LIBUSB_ERROR_OTHER, 107
LIBUSB_ERROR_OVERFLOW, 107
LIBUSB_ERROR_PIPE, 107
LIBUSB_ERROR_TIMEOUT, 107
LIBUSB_RECIPIENT_DEVICE, 107
LIBUSB_RECIPIENT_ENDPOINT, 107
LIBUSB_RECIPIENT_INTERFACE, 107
LIBUSB_RECIPIENT_OTHER, 107
LIBUSB_REQUEST_CLEAR_FEATURE, 106
LIBUSB_REQUEST_GET_CONFIGURATION, 106
LIBUSB_REQUEST_GET_DESCRIPTOR, 106
LIBUSB_REQUEST_GET_INTERFACE, 106
LIBUSB_REQUEST_GET_STATUS, 106
LIBUSB_REQUEST_SET_ADDRESS, 106
LIBUSB_REQUEST_SET_CONFIGURATION, 106
LIBUSB_REQUEST_SET_DESCRIPTOR, 106
LIBUSB_REQUEST_SET_FEATURE, 106
LIBUSB_REQUEST_SET_INTERFACE, 106
LIBUSB_REQUEST_SET_SEL, 107
LIBUSB_REQUEST_SYNCH_FRAME, 106
LIBUSB_REQUEST_TYPE_CLASS, 107
LIBUSB_REQUEST_TYPE_RESERVED, 107
LIBUSB_REQUEST_TYPE_STANDARD, 107
LIBUSB_REQUEST_TYPE_VENDOR, 107
LIBUSB_SET_ISOCH_DELAY, 107
LIBUSB_SUCCESS, 107
LIBUSB_API_VERSION, 106
LIBUSB_CALL, 106
libusb_capability, 107
libusb_error, 107
libusb_error_name, 108
libusb_get_version, 108
libusb_has_capability, 108
libusb_le16_to_cpu, 106
libusb_request_recipient, 107
libusb_request_type, 107
libusb_setlocale, 108
libusb_standard_request, 106
libusb_strerror, 108
missing.c
getenv, 841
missing.h
getenv, 841
SleepEx, 841
mock_capture_usb.c
IN, 695
OUT, 695
mock_dio.c
IN, 697
OUT, 697
mock_usb_xfers.c
IN, 698
OUT, 698
mock_USBDeviceFetchADCConfigBlock
mock_usb_xfers.c, 698
mock_USBDevicePutADCConfigBlock
mock_usb_xfers.c, 698
mock_aiocontbuf_get_data.c
aiocontbuf_get_bulk_data, 693
mock_aiocontbuf_get_data_arduino.c
adc_get_bulk_data, 694
aiocontbuf_get_bulk_data, 694
CloseAllDevices, 694
mock_capture_usb.c
IO_DIRECTION, 695
init_device, 695
InitializeUSBDevice, 696
mock_usb_bulk_transfer, 696
mock_usb_control_transfer, 696
mock_usb_get_config, 696
mock_usb_put_config, 696
mock_usb_request, 696
mock_usb_reset_device, 696
orig_usb_bulk_transfer, 696
orig_usb_control_transfer, 696
orig_usb_get_config, 696
orig_usb_put_config, 696
orig_usb_request, 696
orig_usb_reset_device, 696
outfile, 696
mock_dio.c
IO_DIRECTION, 697
mock_usb_bulk_transfer, 697
mock_usb_control_transfer, 697
save_results, 697
mock_usb_bulk_transfer
mock_capture_usb.c, 696
mock_dio.c, 697
mock_usb_xfers.c, 698
mock_usb_control_transfer
mock_capture_usb.c, 696
mock_dio.c, 697
mock_usb_xfers.c, 698
mock_usb_get_config
mock_capture_usb.c, 696
mock_usb_put_config
mock_capture_usb.c, 696
mock_usb_request
mock_capture_usb.c, 696
mock_usb_reset_device
mock_capture_usb.c, 696
mock_usb_xfers.c, 698
mock_usb_xfers.c
add_devices_fn, 698
AddAllACCESUSBDevices, 698
direction, 698
IO_DIRECTION, 698
KEEP, 698
mock_USBDeviceFetchADCConfigBlock, 698
mock_USBDevicePutADCConfigBlock, 698
mock_usb_bulk_transfer, 698
mock_usb_control_transfer, 698
mock_usb_reset_device, 698
mode
ram_poke_context, 267

msleep
xusb.c, 766
msvc/config.h
DEFAULT_VISIBILITY, 755
ENABLE_LOGGING, 755
HAVE_SIGNAL_H, 755
HAVE_SYS_TYPES_H, 755
OS_WINDOWS, 755
POLL_NFDS_TYPE, 755
mutex
poll_windows.c, 798
mux_settings, 260
ADCChannelsPerGroup, 260
ADCMUXChannels, 260
ADCConfigBlock, 146
defined, 260
myself
devlink_cbarg, 219
mytest.c
BITS_PER_BYTE, 873
DEVICES_REQUIRED, 873
find_dio_96, 873
MAKE_MASK, 873
MASK_BYTES, 873
MAX_DIO_BYTES, 873
MAX_NAME_SIZE, 873
main, 874
PORT_A, 873
PORT_B, 873
PORT_C, 873
show_byte, 873

NO_CONFIG
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.h, 413
NO_DEVICE_FOUND
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 877
NORMAL
linux_usbfs.c, 791
NOT_STARTED
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
NL_GROUP_KERNEL
linux_netlink.c, 790
NONOPTION_P
getopt.c, 762
NULL
getopt1.c, 763
NULL_PATH
testlib.c, 848
NUM_CHANNELS
TestCaseSetup, 275
NUM_PROD_NAMES
AIODeviceTable.c, 470
ndkbuild/jni/AIODeviceTable.c, 474
NUMARGS
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.h, 509
NUMBER_CHANNELS
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
Name
AIODeviceInfo, 161
DeviceProperties, 217
name
ad_gain_pairs, 145
AIODeviceQuery, 162
AIOUSB::USBDeviceBase, 318
DeviceInfo, 216
libusb_testlib_test, 254
option, 262
ProductIDName, 266
usbi_os_backend, 332
NameSize
AIODeviceInfo, 161
nameSize
AIODeviceQuery, 162
DeviceInfo, 216
TestCaseSetup, 275
nano
libusb_version, 258
nargs
devlink_cbarg, 219
nb_driver_names
windows_usb_api_backend, 339
nb_endpoints
windows_device_priv, 338
nb_interfaces
hid_device_priv, 231
ndkbuild/jni/AIOBuf.h
AIO_COUNTS_BUF, 379
AIO_DEFAULT_BUF, 379
AIO_ERROR_BUF, 379
AIO_VOLTS_BUF, 379
ndkbuild/jni/AIOChannelRange.c
BEGIN, 392
END, 392
END_CHANNEL, 392
GAIN, 392
START_CHANNEL, 392
ndkbuild/jni/AIOCommandLine.h
ADCCONFIG_OPT, 408
CHANNEL_OPT, 408
COUNT_OPT, 408
DEBUG_OPT, 408
FILE_OPT, 408
INDEX_NUM, 408
SAMPLE_OPT, 408
SETCAL_OPT, 408
TIMEOUT_OPT, 408
ndkbuild/jni/AIOConfiguration.h
AD_NO_SET_CAL, 414
AD_SCAN_BULKACQUIRE, 413, 414
AD_SCAN_CONTINUOUS, 413, 414
AD_SCAN_GETCHANNEL, 413, 414
AD_SCAN_GETCHANNELV, 413, 414
AD_SCAN_GETSCAN, 413, 414
AD_SCAN_GETSCANV, 413, 414
AD_SET_CAL_AUTO, 414
AD_SET_CAL_MANUAL, 414
AD_SET_CAL_NORMAL, 414
ADCCONIGBLOCK_CONFIG, 413
AIOCONTINUOUSBUF_CONFIG, 413
NO_CONFIG, 413
ndkbuild/jni/AIOContinuousBuffer.h
AIO_CONT_BUF_TYPE_COUNTS, 445
AIO_CONT_BUF_TYPE_VOLTS, 445
ndkbuild/jni/AIOEither.h
aioeither_value_double, 489
aioeither_value_double_t, 489
aioeither_value_int, 489
aioeither_value_int32_t, 489
aioeither_value_longdouble_t, 489
aioeither_value_obj, 489
aioeither_value_string, 489
aioeither_value_uint16_t, 489
aioeither_value_uint32_t, 489
aioeither_value_uint8_t, 489
aioeither_value_unsigned, 489
aioeither_vlaue_int16_t, 489
ndkbuild/jni/AIOTypes.h
ACCES_VENDOR_ID, 543, 545
AD_CAL_MODE_BIP_GROUND, 555

AD_CAL_MODE_GROUND, 555
AD_CAL_MODE_HIGH_REF, 555
AD_CAL_MODE_NORMAL, 555
AD_CAL_MODE_REFERENCE, 555
AD_CONFIG_CAL_MODE, 552
AD_CONFIG_GAIN_CODE, 552
AD_CONFIG_MUX_START_END, 552
AD_CONFIG_OVERSAMPLE, 552
AD_CONFIG_REGISTERS, 555
AD_CONFIG_START_END, 552
AD_CONFIG_START_STOP_CHANNEL_EX, 552
AD_CONFIG_TRIG_COUNT, 552
AD_DIFFERENTIAL_MODE, 552
AD_GAIN_CODE_0_10V, 552
AD_GAIN_CODE_0_1V, 552
AD_GAIN_CODE_0_2V, 552
AD_GAIN_CODE_0_5V, 552
AD_GAIN_CODE_10V, 552
AD_GAIN_CODE_1V, 552
AD_GAIN_CODE_2V, 552
AD_GAIN_CODE_5V, 552
AD_GAIN_CODE_MASK, 551
AD_MAX_CHANNELS, 551
AD_MAX_CONFIG_REGISTERS, 552
AD_MAX_TIMEOUT, 552
AD_MIN_CONFIG_REGISTERS, 552
AD_MIN_TIMEOUT, 552
AD_MUX_CONFIG_REGISTERS, 555
AD_NUM_GAIN_CODE_REGISTERS, 552
AD_NUM_GAIN_CODES, 552
AD_REGISTER_CAL_MODE, 551
AD_REGISTER_GAIN_CODE, 552
AD_REGISTER_MUX_START_END, 551
AD_REGISTER_OVERSAMPLE, 551
AD_REGISTER_START_END, 551
AD_REGISTER_TRIG_COUNT, 551
AD_TRIGGER_CTR0_EXT, 552
AD_TRIGGER_EXTERNAL, 552
AD_TRIGGER_FALLING_EDGE, 552
AD_TRIGGER_SCAN, 552
AD_TRIGGER_TIMER, 552
AD_TRIGGER_VALID_MASK, 552
ADCalMode_begin, 555
ADCalMode_end, 555
ADGainCode_begin, 552
ADGainCode_end, 552
ADRegister_begin, 551
ADRegister_end, 551
AI_16_MAX_COUNTS, 555
AIO_CONTINUE_RUNNING, 555
AIO_PER_CHANNEL, 542
AIO_PER_OVERSAMPLE, 542
AIO_PER_SCANS, 542
AIO_TERMINATE_CALLBACK, 555
AIOCONTINUOUS_BUF_ALLORNONE, 543
AIOCONTINUOUS_BUF_NORMAL, 543
AIOCONTINUOUS_BUF_OVERRIDE, 543
AIOCommandCode_begin, 555
AIOCommandCode_end, 555
AIOContinuousBufMode_begin, 543
AIOContinuousBufMode_end, 543
AIOUSB_ERROR_AIOCOMMANDLINE_HELP, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RANGE, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_INDEX_NUM, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS, 550, 551
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL, 550, 551
AIOUSB_ERROR_BAD_TOKEN_TYPE, 549, 550
AIOUSB_ERROR_DEVICE_NOT_CONNECTED, 549, 550
AIOUSB_ERROR_DEVICE_NOT_FOUND, 549, 550
AIOUSB_ERROR_DIVIDE_BY_ZERO, 549, 550
AIOUSB_ERROR_DUP_NAME, 549, 550
AIOUSB_ERROR_FILE_NOT_FOUND, 549, 550
AIOUSB_ERROR_HANDLE_EOF, 549, 550
AIOUSB_ERROR_INVALID_ADCCONFIG, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_CHANNEL_SETTING, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_DEVICE, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_REGISTER_SETTING, 550, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_SETTING, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE, 549, 551
AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING, 549, 551
AIOUSB_ERROR_INVALID_AIOARGUMENT, 549, 551
AIOUSB_ERROR_INVALID_AIOBUFTYPE, 550, 551
AIOUSB_ERROR_INVALID_AIOCHANNELMASK, 549, 550
AIOUSB_ERROR_INVALID_AIOCMD, 549, 550
AIOUSB_ERROR_INVALID_AIOCONFIGURATION, 549, 551
AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER, 550, 551
AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER_NUM_CHANNELS, 550, 551
AIOUSB_ERROR_INVALID_AIODEVICE_QUERY, 549, 551
AIOUSB_ERROR_INVALID_AIOEITHER, 549, 551
AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION, 549, 550
AIOUSB_ERROR_INVALID_AIOFIFO, 549, 551
AIOUSB_ERROR_INVALID_CALLBACK, 549, 550
AIOUSB_ERROR_INVALID_CALMODE, 549, 551
AIOUSB_ERROR_INVALID_CHANNEL_NUMBER, 549, 551
AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING, 549, 550
AIOUSB_ERROR_INVALID_CONFIG, 549, 551
AIOUSB_ERROR_INVALID_COUNTS, 549, 550
AIOUSB_ERROR_INVALID_COUNTS_CONVERTER, 549, 550
AIOUSB_ERROR_INVALID_DATA, 549, 550
AIOUSB_ERROR_INVALID_DEVICE, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_MUX_CHANNEL_SETTING, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_SETTING, 549, 550
AIOUSB_ERROR_INVALID_DEVICE_STREAM_SETTING, 549, 550
AIOUSB_ERROR_INVALID_DIOBUF, 549, 551
AIOUSB_ERROR_INVALID_GAINCODE, 549, 551
AIOUSB_ERROR_INVALID_INDEX, 549, 550
AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE, 550, 551

AIOUSB_ERROR_INVALID_MEMORY, 549, 550
AIOUSB_ERROR_INVALID_MUTEX, 549, 550
AIOUSB_ERROR_INVALID_PARAMETER, 549, 550
AIOUSB_ERROR_INVALID_THREAD, 549, 550
AIOUSB_ERROR_INVALID_TIMEOUT, 549, 550
AIOUSB_ERROR_INVALID_USBDEVICE, 549, 550
AIOUSB_ERROR_INVALID_VOLTAGES, 549, 550
AIOUSB_ERROR_LIBUSB, 550, 551
AIOUSB_ERROR_NOT_ENOUGH_MEMORY, 549, 550
AIOUSB_ERROR_NOT_INIT, 549, 550
AIOUSB_ERROR_NOT_SUPPORTED, 549, 550
AIOUSB_ERROR_OPEN_FAILED, 549, 550
AIOUSB_ERROR_TIMEOUT, 549, 550
AIOUSB_ERROR_USB_INIT, 549, 550
AIOUSB_ERROR_USBDEVICE_NOT_FOUND, 549, 550
AIOUSB_FALSE, 543
AIOUSB_FIFO_COPY_ERROR, 550, 551
AIOUSB_MAX_NAME_SIZE, 555
AIOUSB_SUCCESS, 549, 550
AIOUSB_TRUE, 543
AUR_ADC_GET_CONFIG, 553, 554
AUR_ADC_IMMEDIATE, 553, 554
AUR_ADC_SET_CONFIG, 553, 554
AUR_CTR_COS_BULK_ABORT, 553, 554
AUR_CTR_COS_BULK_GATE2, 553, 554
AUR_CTR_LOAD, 553, 554
AUR_CTR_MODE, 553, 554
AUR_CTR_MODELOAD, 553, 554
AUR_CTR_PUR_EVCT, 553, 554
AUR_CTR_PUR_FIRST, 553, 554
AUR_CTR_PUR_MFRQ, 553, 554
AUR_CTR_PUR_MPUL, 553, 554
AUR_CTR_PUR_OFRQ, 553, 554
AUR_CTR_READ, 553, 554
AUR_CTR_READALL, 553, 554
AUR_CTR_READLATCHED, 553, 554
AUR_CTR_SELGATE, 553, 554
AUR_DAC_CONTROL, 553, 554
AUR_DAC_DATAPTR, 553, 554
AUR_DAC_DIVISOR, 553, 554
AUR_DAC_IMMEDIATE, 553, 554
AUR_DAC_RANGE, 553, 554
AUR_DIO_CONFIG, 553, 554
AUR_DIO_CONFIG_QUERY, 553, 554
AUR_DIO_READ, 553, 554
AUR_DIO_SETCLOCKS, 553, 554
AUR_DIO_SPI_READ, 553, 554
AUR_DIO_SPI_WRITE, 553, 554
AUR_DIO_STREAM_OPEN_INPUT, 553, 554
AUR_DIO_STREAM_OPEN_OUTPUT, 553, 554
AUR_DIO_WDG16_DEPREC, 553, 554
AUR_DIO_WRITE, 553
AUR_EEPROM_READ, 553, 554
AUR_EEPROM_WRITE, 553, 554
AUR_FLASH_READWRITE, 553, 554
AUR_GEN_ABORT_AND_CLEAR, 553, 554
AUR_GEN_CLEAR_FIFO, 553, 554
AUR_GEN_CLEAR_FIFO_NEXT, 553, 554
AUR_GEN_CLEAR_FIFO_WAIT, 553, 554
AUR_GEN_STREAM_STATUS, 553, 554
AUR_LOAD_BULK_CALIBRATION_BLOCK, 553, 554
AUR_OFFLINE_READWRITE, 553, 554
AUR_PROBE_CALFEATURE, 553, 554
AUR_READBACK_GLOBAL_STATE, 553, 554
AUR_SAVE_GLOBAL_STATE, 553, 554
AUR_SELF_TEST_1, 553, 554
AUR_START_ACQUIRING_BLOCK, 553, 554
AUR_WDG, 553, 554
AUR_WDG_STATUS, 553, 554
BITS_PER_BYTE, 555
CAL_TABLE_WORDS, 555
CLEAR_FIFO_METHOD_AUTO, 548
CLEAR_FIFO_METHOD_IMMEDIATE, 548
CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT, 548
CLEAR_FIFO_METHOD_NOW, 548
CLEAR_FIFO_METHOD_WAIT, 548
COUNTER_NUM_MODES, 555
COUNTERS_PER_BLOCK, 555
CYPRESS_DESC_PARAMS, 555
CYPRESS_GET_DESC, 553, 554
CYPRESS_MAX_DESC_SIZE, 555
DAC_RANGE_0_10V, 548
DAC_RANGE_0_5V, 548
DAC_RANGE_10V, 548
DAC_RANGE_5V, 548
DAC_RESET, 555
DACRange_begin, 548
DACRange_end, 548
diFirst, 547
diNone, 547
diOnly, 547
EEPROM_CUSTOM_BASE_ADDRESS, 555
EEPROM_CUSTOM_MAX_ADDRESS, 555
EEPROM_CUSTOM_MIN_ADDRESS, 555
EEPROM_SERIAL_NUMBER_ADDRESS, 555
FIFO_Method_begin, 548
FIFO_Method_end, 548
GENERIC_DOSOMETHING_PLACEHOLDER, 555
INVALID_OBJECT, 542
JOINED, 542
MAX_IMM_ADCS, 555
MAX_USB_DEVICES, 543
NOT_STARTED, 542
PICO_DIO16RO8, 544, 546
ProductIDS_begin, 543, 545
ProductIDS_end, 545, 547
RUNNING, 542
RUNNING_OR_WITH_DATA, 542
ResultCode_begin, 549, 550
ResultCode_end, 550, 551
TERMINATED, 542
TERMINATED_OVERRUN, 542
TERMINATING, 542
THREAD_STATUS_begin, 542
THREAD_STATUS_end, 542
USB_AI12_128, 544, 546
USB_AI12_128A, 544, 546
USB_AI12_128E, 544, 546
USB_AI12_16, 544, 546
USB_AI12_16A, 544, 546
USB_AI12_16E, 544, 546
USB_AI12_32, 544, 546
USB_AI12_32A, 544, 546
USB_AI12_32E, 544, 546
USB_AI12_64, 544, 546
USB_AI12_64A, 544, 546
USB_AI12_64E, 544, 546
USB_AI12_64M, 544, 546
USB_AI12_64MA, 544, 546
USB_AI12_64ME, 544, 546
USB_AI12_96, 544, 546
USB_AI12_96A, 544, 546
USB_AI12_96E, 544, 546
USB_AI16_128A, 544, 546
USB_AI16_128E, 544, 546
USB_AI16_16A, 544, 546
USB_AI16_16E, 544, 546
USB_AI16_32A, 544, 546
USB_AI16_32E, 544, 546

USB_AI16_64A, 544, 546
USB_AI16_64E, 544, 546
USB_AI16_64MA, 544, 546
USB_AI16_64ME, 544, 546
USB_AI16_96A, 544, 546
USB_AI16_96E, 544, 546
USB_AIO12_128, 545, 547
USB_AIO12_128A, 545, 547
USB_AIO12_128E, 545, 547
USB_AIO12_16, 545, 547
USB_AIO12_16A, 545, 547
USB_AIO12_16E, 545, 547
USB_AIO12_32, 545, 547
USB_AIO12_32A, 545, 547
USB_AIO12_32E, 545, 547
USB_AIO12_64, 545, 547
USB_AIO12_64A, 545, 547
USB_AIO12_64E, 545, 547
USB_AIO12_64M, 545, 547
USB_AIO12_64MA, 545, 547
USB_AIO12_64ME, 545, 547
USB_AIO12_96, 545, 547
USB_AIO12_96A, 545, 547
USB_AIO12_96E, 545, 547
USB_AIO16_128A, 545, 547
USB_AIO16_128E, 545, 547
USB_AIO16_16A, 545, 547
USB_AIO16_16E, 545, 547
USB_AIO16_32A, 545, 547
USB_AIO16_32E, 545, 547
USB_AIO16_64A, 545, 547
USB_AIO16_64E, 545, 547
USB_AIO16_64MA, 545, 547
USB_AIO16_64ME, 545, 547
USB_AIO16_96A, 545, 547
USB_AIO16_96E, 545, 547
USB_AO12_12, 545, 547
USB_AO12_12A, 545, 547
USB_AO12_16, 545, 547
USB_AO12_16A, 544, 547
USB_AO12_4, 545, 547
USB_AO12_4A, 545, 547
USB_AO12_8, 545, 547
USB_AO12_8A, 545, 547
USB_AO16_12, 544, 547
USB_AO16_12A, 544, 546
USB_AO16_16, 544, 546
USB_AO16_16A, 544, 546
USB_AO16_4, 544, 547
USB_AO16_4A, 544, 547
USB_AO16_8, 544, 547
USB_AO16_8A, 544, 547
USB_AO_ARB1, 544, 546
USB_BULK_READ_ENDPOINT, 555
USB_BULK_WRITE_ENDPOINT, 555
USB_CTR_15, 544, 546
USB_DA12_8A, 543, 545
USB_DA12_8A_REV_A, 543, 545
USB_DA12_8E, 543, 545
USB_DI16A, 543, 545
USB_DI16A_REV_A1, 543, 545
USB_DI16A_REV_A2, 543, 545
USB_DIO16RO8, 544, 546
USB_DIO24_CTR6, 543, 545
USB_DIO24DO12, 544, 546
USB_DIO48DO24, 544, 546
USB_DIO_16A, 543, 545
USB_DIO_16H, 543, 545
USB_DIO_32, 543, 545
USB_DIO_32I, 543, 545
USB_DIO_48, 543, 545
USB_DIO_96, 543, 545
USB_DO16A, 543, 545
USB_DO16A_REV_A1, 543, 545
USB_DO24, 544, 546
USB_IDIO_16, 544, 546
USB_IDIO_4, 544, 546
USB_IDIO_8, 544, 546
USB_IDO_16, 544, 546
USB_II_16, 543, 546
USB_II_16_OLD, 544, 546
USB_II_8, 543, 546
USB_II_8_OLD, 544, 546
USB_IIRO4_2SM, 544, 546
USB_IIRO4_COM, 544, 546
USB_IIRO_16, 543, 546
USB_IIRO_4, 543, 546
USB_IIRO_8, 543, 546
USB_READ_FROM_DEVICE, 555
USB_RO_16, 543, 546
USB_WRITE_TO_DEVICE, 555
USBP_II8IDO4A, 544, 546
VENDOR_REQUEST_begin, 553
VENDOR_REQUEST_end, 553, 554
WITH_DATA, 542
ndkbuild/jni/AIOUSB_Log.h
AIODEBUG_LEVEL, 650
AIODEFAULT_LOG_LEVEL, 650
AIODEVEL_LEVEL, 650
AIOERROR_LEVEL, 650
AIOFATAL_LEVEL, 650
AIOINFO_LEVEL, 650
AIOWARN_LEVEL, 650
ndkbuild/jni/AIOUSB_Properties.h
BASIC, 656, 657
JSON, 656, 657
TERSE, 656, 657
YAML, 657
ndkbuild/jni/AIOUSB_WDG.h
AIOUSB_WDG_READ_INDEX, 664
AIOUSB_WDG_READ_VALUE, 664
WDGVals_begin, 664
WDGVals_end, 664
ndkbuild/jni/ADCConfigBlock.c
ADC_VerifyAndCorrectConfigBlock, 365
ADCConfigBlockCopy, 364
ADCConfigBlockGetAIOUSBDevice, 364
ADCConfigBlockGetCalMode, 365
ADCConfigBlockGetClockRate, 366
ADCConfigBlockGetDebug, 364
ADCConfigBlockGetEndChannel, 365
ADCConfigBlockGetGainCode, 365
ADCConfigBlockGetJSONValueOrDefault, 366
ADCConfigBlockGetJSONValueOrInt, 366
ADCConfigBlockGetOversample, 365
ADCConfigBlockGetSize, 364
ADCConfigBlockGetStartChannel, 365
ADCConfigBlockGetTesting, 364
ADCConfigBlockGetTimeout, 365
ADCConfigBlockGetTriggerMode, 365
ADCConfigBlockInit, 364
ADCConfigBlockInitForCounterScan, 364
ADCConfigBlockInitializeDefault, 364
ADCConfigBlockInitializeFromAIOUSBDevice, 364
ADCConfigBlockSetAIOUSBDevice, 364
ADCConfigBlockSetAllGainCodeAndDiffMode, 365
ADCConfigBlockSetCalMode, 365
ADCConfigBlockSetChannelRange, 365
ADCConfigBlockSetClockRate, 366
ADCConfigBlockSetDebug, 364
ADCConfigBlockSetDevice, 364
ADCConfigBlockSetDifferentialMode, 365
ADCConfigBlockSetEndChannel, 365
ADCConfigBlockSetGainCode, 365

ADCConfigBlockSetOversample, 365
ADCConfigBlockSetRangeSingle, 364
ADCConfigBlockSetReference, 365
ADCConfigBlockSetRegister, 364
ADCConfigBlockSetScanAllChannels, 366
ADCConfigBlockSetScanRange, 365
ADCConfigBlockSetSize, 364
ADCConfigBlockSetStartChannel, 365
ADCConfigBlockSetTesting, 364
ADCConfigBlockSetTimeout, 365
ADCConfigBlockSetTriggerEdge, 365
ADCConfigBlockSetTriggerMode, 365
ADCConfigBlockSetTriggerReference, 366
ADCConfigBlockToJSON, 366
ADCConfigBlockToYAML, 365
DeleteADCConfigBlock, 364
get_cal_mode, 365
get_gain_code, 365
is_all_digits, 366
NewADCConfigBlockFromJSON, 366
ndkbuild/jni/ADCConfigBlock.h
ADC_VerifyAndCorrectConfigBlock, 372
ADCConfigBlock, 372
ADCConfigBlockCopy, 374
ADCConfigBlockGetAIOUSBDevice, 374
ADCConfigBlockGetCalMode, 373
ADCConfigBlockGetClockRate, 373
ADCConfigBlockGetDebug, 374
ADCConfigBlockGetEndChannel, 373
ADCConfigBlockGetGainCode, 372
ADCConfigBlockGetOversample, 373
ADCConfigBlockGetSize, 374
ADCConfigBlockGetStartChannel, 373
ADCConfigBlockGetTesting, 374
ADCConfigBlockGetTimeout, 373
ADCConfigBlockGetTriggerMode, 373
ADCConfigBlockInit, 372
ADCConfigBlockInitForCounterScan, 372
ADCConfigBlockInitializeDefault, 372
ADCConfigBlockInitializeFromAIOUSBDevice, 374
ADCConfigBlockSetAIOUSBDevice, 374
ADCConfigBlockSetAllGainCodeAndDiffMode, 372
ADCConfigBlockSetCalMode, 373
ADCConfigBlockSetChannelRange, 373
ADCConfigBlockSetClockRate, 372
ADCConfigBlockSetDebug, 374
ADCConfigBlockSetDevice, 374
ADCConfigBlockSetDifferentialMode, 374
ADCConfigBlockSetEndChannel, 373
ADCConfigBlockSetGainCode, 372
ADCConfigBlockSetOversample, 373
ADCConfigBlockSetRangeSingle, 374
ADCConfigBlockSetReference, 373
ADCConfigBlockSetRegister, 372
ADCConfigBlockSetScanRange, 373
ADCConfigBlockSetSize, 374
ADCConfigBlockSetStartChannel, 373
ADCConfigBlockSetTesting, 374
ADCConfigBlockSetTimeout, 373
ADCConfigBlockSetTriggerEdge, 374
ADCConfigBlockSetTriggerMode, 373
ADCConfigBlockToJSON, 374
ADCConfigBlockToYAML, 373
ADCMuxSettings, 372
ADConfigBlock, 372
AIOUSBDevice, 372
DeleteADCConfigBlock, 374
is_all_digits, 374
NewADCConfigBlockFromJSON, 374
ndkbuild/jni/AIOBuf.c
AIOBufGetIterator, 376
AIOBufGetRaw, 376
AIOBufGetSize, 376
AIOBufGetTotalSize, 376
AIOBufGetType, 376
AIOBufGetTypeSize, 376
AIOBufIteratorGetValue, 376
AIOBufIteratorIsValid, 376
AIOBufIteratorNext, 376
AIOBufRead, 376
AIOBufWrite, 376
DeleteAIOBuf, 376
NewAIOBuf, 376
ndkbuild/jni/AIOBuf.h
AIOBuf, 379
AIOBufGetIterator, 379
AIOBufGetRaw, 379
AIOBufGetSize, 379
AIOBufGetType, 379
AIOBufIterator, 379
AIOBufIteratorGetValue, 379
AIOBufIteratorIsValid, 379
AIOBufIteratorNext, 380
AIOBufRead, 379
AIOBufType, 379
AIOBufWrite, 379
DeleteAIOBuf, 379
NewAIOBuf, 379
ndkbuild/jni/AIOChannelMask.c
AIOChannelMaskGetMask, 384
AIOChannelMaskGetMaskAtIndex, 383
AIOChannelMaskGetSize, 383
AIOChannelMaskIndices, 383
AIOChannelMaskNextIndex, 383
AIOChannelMaskNumberChannels, 383
AIOChannelMaskNumberSignals, 384
AIOChannelMaskSetMaskAtIndex, 383
AIOChannelMaskSetMaskFromInt, 383
AIOChannelMaskSetMaskFromStr, 384
AIOChannelMaskToString, 384
AIOChannelMaskToStringAtIndex, 384
DeleteAIOChannelMask, 383
NewAIOChannelMask, 383
NewAIOChannelMaskFromChr, 384
NewAIOChannelMaskFromStr, 384
ndkbuild/jni/AIOChannelMask.h
AIOChannelMaskGetMask, 389
AIOChannelMaskGetMaskAtIndex, 389
AIOChannelMaskGetSize, 390
AIOChannelMaskIndices, 390
AIOChannelMaskNextIndex, 390
AIOChannelMaskNumberChannels, 389
AIOChannelMaskNumberSignals, 390
AIOChannelMaskSetMaskAtIndex, 390
AIOChannelMaskSetMaskFromInt, 390
AIOChannelMaskSetMaskFromStr, 390
AIOChannelMaskToString, 389
AIOChannelMaskToStringAtIndex, 389
aio_channel_obj, 388
BIT_LENGTH, 388
DeleteAIOChannelMask, 388
NewAIOChannelMask, 388
NewAIOChannelMaskFromChr, 389
NewAIOChannelMaskFromStr, 388
ndkbuild/jni/AIOChannelRange.c
AIOChannelRangeGetEnd, 392
AIOChannelRangeGetGain, 392
AIOChannelRangeGetStart, 392
AIOChannelRangeToStr, 392
aio_channel_range_error, 392
DeleteAIOChannelRange, 392
lookup_voltage_range, 392
NewAIOChannelRangeFromStr, 392
STATE, 392

ndkbuild/jni/AIOChannelRange.h
AIOChannelRange, 394
AIOChannelRangeGetEnd, 394
AIOChannelRangeGetGain, 394
AIOChannelRangeGetStart, 394
AIOChannelRangeToStr, 394
aio_channel_range_error, 394
DeleteAIOChannelRange, 394
NewAIOChannelRangeFromStr, 394
ndkbuild/jni/AIOCmd.c
DeleteAIOCmd, 395
NewAIOCmd, 395
NewAIOCmdFromJSON, 395
ndkbuild/jni/AIOCmd.h
AIOCmd, 396
DeleteAIOCmd, 396
NewAIOCmd, 396
NewAIOCmdFromJSON, 396
ndkbuild/jni/AIOCommandLine.c
AIOCommandLineListDevices, 400
AIOCommandLineOptionsListDevices, 401
AIOCommandLineOptionsOverrideADCConfigBlock, 402
AIOCommandLineOverrideADCConfigBlock, 402
AIOGetChannelRange, 402
AIOPrintUsage, 400
AIOProcessCmdline, 400
AIOProcessCommandLine, 400
DeleteAIOCommandLineOptions, 400
NewAIOCommandLineOptionsFromDefaultOptions, 400
NewDefaultAIOCommandLineOptions, 400
opterr, 402
optind, 402
ndkbuild/jni/AIOCommandLine.h
AIOChannelRangeTmp, 407
AIOCommandLineListDevices, 409
AIOCommandLineOptions, 407
AIOCommandLineOptionsListDevices, 409
AIOCommandLineOptionsOverrideADCConfigBlock, 409
AIOCommandLineOverrideADCConfigBlock, 409
AIOGetChannelRange, 409
AIOPrintUsage, 409
AIOProcessCmdline, 408
AIOProcessCommandLine, 408
CNTS, 407
DUMP, 407
DeleteAIOCommandLineOptions, 409
DeviceEnum, 408
JCONF, 407
NewAIOCommandLineOptionsFromDefaultOptions, 408
NewDefaultAIOCommandLineOptions, 408
REPEAT, 407
ndkbuild/jni/AIOConfiguration.c
AIOArgumentInitialize, 411
AIOConfigurationInitialize, 411
AIOConfigurationSetDebug, 411
AIOConfigurationSetTimeout, 411
DeleteAIOConfiguration, 411
NewAIOConfiguration, 411
ndkbuild/jni/AIOConfiguration.h
ADCConfiguration, 413
ADCScanType, 413
ADCSetCalFunction, 414
AIOArgumentInitialize, 414
AIOConfiguration, 413
AIOConfigurationInitialize, 414
AIOConfigurationSetDebug, 414
AIOConfigurationSetTimeout, 414
AIOContinousBufConfiguration, 413
ConfigurationType, 413
NewAIOConfiguration, 414
ndkbuild/jni/AIOContinuousBuffer.c
AIOContinuousBuf_BufSizeForCounts, 429
AIOContinuousBuf_GetDeviceIndex, 434
AIOContinuousBuf_GetOverSample, 434
AIOContinuousBuf_InitConfiguration, 429
AIOContinuousBuf_NumberChannels, 431
AIOContinuousBuf_NumberSignals, 431
AIOContinuousBuf_SaveConfig, 433
AIOContinuousBuf_SendPreConfig, 429
AIOContinuousBuf_SetAllGainCodeAndDiffMode, 434
AIOContinuousBuf_SetCallback, 429
AIOContinuousBuf_SetChannelRange, 433
AIOContinuousBuf_SetChannelRangeGain, 433
AIOContinuousBuf_SetDeviceIndex, 433
AIOContinuousBuf_SetDiscardFirstSample, 434
AIOContinuousBuf_SetOversample, 434
AIOContinuousBuf_SetStartAndEndChannel, 433
AIOContinuousBuf_SetTesting, 433
AIOContinuousBufCallbackStart, 432
AIOContinuousBufCallbackStartCallbackWith-AcquisitionFunction, 432
AIOContinuousBufCleanup, 432
AIOContinuousBufCountScansAvailable, 430
AIOContinuousBufCountsAvailable, 430
AIOContinuousBufEnd, 433
AIOContinuousBufForceTerminateAcqusition, 428
AIOContinuousBufForceTerminateAcqusitionOverrun, 428
AIOContinuousBufGetADCConfigBlock, 429
AIOContinuousBufGetBaseSize, 429
AIOContinuousBufGetBufferSize, 429
AIOContinuousBufGetCallback, 431
AIOContinuousBufGetClock, 431
AIOContinuousBufGetDataAvailable, 430
AIOContinuousBufGetDebug, 433
AIOContinuousBufGetDeviceIndex, 434
AIOContinuousBufGetExitCode, 430
AIOContinuousBufGetNumberChannels, 428
AIOContinuousBufGetNumberOfScansToRead, 430
AIOContinuousBufGetNumberSamplesPerScan, 432
AIOContinuousBufGetNumberScans, 429
AIOContinuousBufGetOversample, 434
AIOContinuousBufGetReadPosition, 430
AIOContinuousBufGetRemainingSize, 430
AIOContinuousBufGetRunStatus, 430
AIOContinuousBufGetScansRead, 430
AIOContinuousBufGetSize, 430
AIOContinuousBufGetSizeNumElements, 430
AIOContinuousBufGetStatus, 430
AIOContinuousBufGetStreamingBlockSize, 429
AIOContinuousBufGetTesting, 433
AIOContinuousBufGetTimeout, 434
AIOContinuousBufGetTotalSamplesExpected, 432
AIOContinuousBufGetUnitSize, 429
AIOContinuousBufGetWritePosition, 430
AIOContinuousBufInitADCConfigBlock, 429
AIOContinuousBufInitConfiguration, 429
AIOContinuousBufInitiateCallbackAcquisition, 432
AIOContinuousBufLoadCounters, 432
AIOContinuousBufLock, 433
AIOContinuousBufNumberChannels, 431
AIOContinuousBufNumberSamplesAvailable, 432
AIOContinuousBufNumberSignals, 431
AIOContinuousBufNumberWriteSamplesRemaining, 432
AIOContinuousBufPending, 430
AIOContinuousBufPopN, 429
AIOContinuousBufPreSetup, 432
AIOContinuousBufPushN, 429

AIOContinuousBufRead, 433
AIOContinuousBufReadCompleteScanCounts, 431
AIOContinuousBufReadIntegerNumberOfScans, 430
AIOContinuousBufReadIntegerScanCounts, 430
AIOContinuousBufReadNSamples, 432
AIOContinuousBufReadSingle, 430
AIOContinuousBufReset, 429
AIOContinuousBufResetDevice, 433
AIOContinuousBufSaveConfig, 433
AIOContinuousBufSendPreConfig, 429
AIOContinuousBufSetAllGainCodeAndDiffMode, 434
AIOContinuousBufSetBaseSize, 429
AIOContinuousBufSetCallback, 429
AIOContinuousBufSetChannelMask, 431
AIOContinuousBufSetChannelRange, 434
AIOContinuousBufSetClock, 431
AIOContinuousBufSetCountsBuffer, 429
AIOContinuousBufSetDebug, 433
AIOContinuousBufSetDefaultModeForCounterScan, 433
AIOContinuousBufSetDeviceIndex, 433
AIOContinuousBufSetDiscardFirstSample, 434
AIOContinuousBufSetNumberChannels, 429
AIOContinuousBufSetNumberOfScansToRead, 430
AIOContinuousBufSetNumberScans, 429
AIOContinuousBufSetOverSample, 434
AIOContinuousBufSetOversample, 434
AIOContinuousBufSetStartAndEndChannel, 433
AIOContinuousBufSetStreamingBlockSize, 429
AIOContinuousBufSetTesting, 433
AIOContinuousBufSetTimeout, 434
AIOContinuousBufSetUnitSize, 429
AIOContinuousBufSetVoltsBuffer, 429
AIOContinuousBufSimpleSetupConfig, 433
AIOContinuousBufStart, 431
AIOContinuousBufStopAcquisition, 431
AIOContinuousBufToJSON, 434
AIOContinuousBufUnlock, 433
AIOContinuousBufWrite, 431
AIOContinuousBufWriteCounts, 432
BaseSizeRange, 434
continue_running, 432
ConvertCountsToVoltsFunction, 428
DeleteAIOContinuousBuf, 429
GetJSONValueOrDefault, 434
NewAIOContinuousBuf, 428
NewAIOContinuousBufForCounts, 428
NewAIOContinuousBufForVolts, 429
NewAIOContinuousBufFromJSON, 434
number_to_read, 432
RangeValueLookup, 428
RawCountsWorkFunction, 428
ResetCounters, 432
SetConfig, 432
StartStreaming, 432
TrueFalse, 434
ndkbuild/jni/AIOContinuousBuffer.h
AIOContinuousBuf, 445
AIOContinuousBufCallbackStart, 448
AIOContinuousBufCallbackStartCallbackWith-AcquisitionFunction, 449
AIOContinuousBufCleanup, 451
AIOContinuousBufCountScansAvailable, 450
AIOContinuousBufEnd, 450
AIOContinuousBufGetADCConfigBlock, 446
AIOContinuousBufGetBaseSize, 446
AIOContinuousBufGetBufferSize, 446
AIOContinuousBufGetCallback, 446
AIOContinuousBufGetClock, 450
AIOContinuousBufGetDebug, 448
AIOContinuousBufGetDeviceIndex, 446
AIOContinuousBufGetExitCode, 449
AIOContinuousBufGetNumberChannels, 446
AIOContinuousBufGetNumberSamplesPerScan, 448
AIOContinuousBufGetNumberScans, 448
AIOContinuousBufGetOversample, 446
AIOContinuousBufGetReadPosition, 449
AIOContinuousBufGetRemainingSize, 449
AIOContinuousBufGetRunStatus, 449
AIOContinuousBufGetScansRead, 449
AIOContinuousBufGetStatus, 449
AIOContinuousBufGetStreamingBlockSize, 446
AIOContinuousBufGetTesting, 446
AIOContinuousBufGetTimeout, 448
AIOContinuousBufGetTotalSamplesExpected, 448
AIOContinuousBufGetUnitSize, 446
AIOContinuousBufGetWritePosition, 449
AIOContinuousBufInitADCConfigBlock, 446
AIOContinuousBufInitConfiguration, 445
AIOContinuousBufInitiateCallbackAcquisition, 449
AIOContinuousBufLock, 448
AIOContinuousBufNumberChannels, 446
AIOContinuousBufNumberSamplesAvailable, 448
AIOContinuousBufNumberSignals, 448
AIOContinuousBufNumberWriteSamplesRemaining, 448
AIOContinuousBufPending, 449
AIOContinuousBufPopN, 448
AIOContinuousBufPushN, 448
AIOContinuousBufRead, 450
AIOContinuousBufReadCompleteScanCounts, 450
AIOContinuousBufReadIntegerNumberOfScans, 450
AIOContinuousBufReadIntegerScanCounts, 449
AIOContinuousBufReset, 448
AIOContinuousBufResetDevice, 448
AIOContinuousBufSaveConfig, 448
AIOContinuousBufSendPreConfig, 446
AIOContinuousBufSetAllGainCodeAndDiffMode, 446
AIOContinuousBufSetBaseSize, 446
AIOContinuousBufSetCallback, 446
AIOContinuousBufSetChannelMask, 447
AIOContinuousBufSetChannelRange, 448
AIOContinuousBufSetClock, 450
AIOContinuousBufSetCountsBuffer, 450
AIOContinuousBufSetDebug, 448
AIOContinuousBufSetDeviceIndex, 448
AIOContinuousBufSetDiscardFirstSample, 446
AIOContinuousBufSetNumberChannels, 446
AIOContinuousBufSetNumberScans, 448
AIOContinuousBufSetOversample, 446
AIOContinuousBufSetStartAndEndChannel, 446
AIOContinuousBufSetStreamingBlockSize, 446
AIOContinuousBufSetTesting, 446
AIOContinuousBufSetTimeout, 448
AIOContinuousBufSetUnitSize, 446
AIOContinuousBufSetVoltsBuffer, 450
AIOContinuousBufSimpleSetupConfig, 450
AIOContinuousBufStopAcquisition, 449
AIOContinuousBufToJSON, 451
AIOContinuousBufUnlock, 448
AIOContinuousBufWrite, 450
AIOContinuousBufWriteCounts, 451
AIOUSB_WorkFn, 445
DeleteAIOContinuousBuf, 445
NewAIOContinuousBuf, 445
NewAIOContinuousBufForCounts, 445
NewAIOContinuousBufForVolts, 445
NewAIOContinuousBufFromJSON, 451
ROOTCLOCK, 445
ndkbuild/jni/AIOCountsConverter.c
AIOCountsConverterConvert, 453
AIOCountsConverterConvertAllAvailableScans, 453
AIOCountsConverterConvertFifo, 453
AIOCountsConverterConvertNScans, 453

AIOCountsConverterReset, 453
Convert, 453
default_out, 453
DeleteAIOCountsConverter, 453
DeleteAIOGainRange, 453
enhanced_out, 453
NewAIOCountsConverter, 453
NewAIOCountsConverterWithBuffer, 453
NewAIOCountsConverterWithScanLimiter, 453
NewAIOGainRangeFromADCConfigBlock, 453
ndkbuild/jni/AIOCountsConverter.h
AIOCountsConverter, 455
AIOCountsConverterConvert, 455
AIOCountsConverterConvertAllAvailableScans, 455
AIOCountsConverterConvertFifo, 455
AIOCountsConverterConvertNScans, 455
AIOCountsConverterReset, 455
DeleteAIOCountsConverter, 455
DeleteAIOGainRange, 456
NewAIOCountsConverter, 455
NewAIOCountsConverterFromAIOContinuousBuf, 455
NewAIOCountsConverterWithBuffer, 455
NewAIOCountsConverterWithScanLimiter, 455
NewAIOGainRangeFromADCConfigBlock, 456
ndkbuild/jni/AIODeviceInfo.c
AIODeviceInfoGet, 457
AIODeviceInfoGetCounters, 457
AIODeviceInfoGetDIOBytes, 457
AIODeviceInfoGetName, 457
DeleteAIODeviceInfo, 457
NewAIODeviceInfo, 457
ndkbuild/jni/AIODeviceInfo.h
AIODeviceInfo, 458
AIODeviceInfoGet, 458
AIODeviceInfoGetCounters, 458
AIODeviceInfoGetDIOBytes, 458
AIODeviceInfoGetName, 458
DeleteAIODeviceInfo, 458
NewAIODeviceInfo, 458
ndkbuild/jni/AIODeviceQuery.c
AIODeviceQueryGetIndex, 462
AIODeviceQueryGetName, 464
AIODeviceQueryGetNumCounters, 464
AIODeviceQueryGetNumDIOBytes, 464
AIODeviceQueryGetProductID, 462
AIODeviceQueryNameSize, 464
AIODeviceQueryToRepr, 462
AIODeviceQueryToStr, 462
DeleteAIODeviceQuery, 462
NewAIODeviceQuery, 462
ndkbuild/jni/AIODeviceQuery.h
AIODeviceQuery, 468
AIODeviceQueryGetIndex, 469
AIODeviceQueryGetName, 469
AIODeviceQueryGetNumCounters, 469
AIODeviceQueryGetNumDIOBytes, 469
AIODeviceQueryGetProductID, 468
AIODeviceQueryNameSize, 468
AIODeviceQueryToRepr, 468
AIODeviceQueryToStr, 468
DeleteAIODeviceQuery, 468
NewAIODeviceQuery, 468
ndkbuild/jni/AIODeviceTable.c
AIODeviceTableAddDeviceToDeviceTable, 476
AIODeviceTableClearDevices, 476
AIODeviceTableGetDeviceAtIndex, 475
AIODeviceTableGetUSBDeviceAtIndex, 475
AIODeviceTableInit, 474
AIODeviceTablePopulateTable, 476
AIODeviceTablePopulateTableTest, 476
AIODeviceTableSetDeviceID, 476
AIOUSB_Cleanup, 474
AIOUSB_EnsureOpen, 475
AIOUSB_Exit, 476
AIOUSB_GetAllDevices, 476
AIOUSB_Init, 476
AIOUSB_InitTest, 474
AIOUSB_IsInit, 474
AIOUSB_Reset, 476
AIOUSB_SetInit, 474
AIOUSBGetError, 476
aiousbInit, 476
ClearAIODeviceTable, 476
ClearDevices, 476
CloseAllDevices, 476
deviceTable, 476
GetDevices, 475
GetSafeDeviceName, 475
ProductIDToName, 474
ProductNameToID, 474
QueryDeviceInfo, 474
ndkbuild/jni/AIODeviceTable.h
AIODeviceTableAddDeviceToDeviceTable, 481
AIODeviceTableClearDevices, 481
AIODeviceTableGetDeviceAtIndex, 481
AIODeviceTableGetUSBDeviceAtIndex, 481
AIODeviceTableInit, 483
AIODeviceTablePopulateTable, 481
AIODeviceTablePopulateTableTest, 481
AIOUSB_EnsureOpen, 483
AIOUSB_Exit, 483
AIOUSB_GetAllDevices, 484
AIOUSB_Init, 483
AIOUSB_IsInit, 483
AIOUSB_Reset, 483
AIOUSBGetError, 484
ClearAIODeviceTable, 483
ClearDevices, 481
CloseAllDevices, 483
deviceTable, 484
GetDevices, 482
GetSafeDeviceName, 482
ProductIDToName, 482
ProductNameToID, 482
QueryDeviceInfo, 482
ndkbuild/jni/AIOEither.c
AIOEitherClear, 486
AIOEitherGetLeft, 486
AIOEitherGetRight, 486
AIOEitherHasError, 486
AIOEitherSetLeft, 486
AIOEitherSetRight, 486
AIOEitherToAIONumber, 486
AIOEitherToAIORetType, 486
AIOEitherToDouble, 486
AIOEitherToInt, 486
AIOEitherToShort, 486
AIOEitherToString, 486
AIOEitherToUnsigned, 486
LOOKUP, 485
ndkbuild/jni/AIOEither.h
AIO_EITHER_TYPE, 489
AIO_ERROR_VALUE, 489
AIOEither, 489
AIOEitherClear, 490
AIOEitherGetLeft, 490
AIOEitherGetRight, 490
AIOEitherHasError, 490
AIOEitherSetLeft, 490
AIOEitherSetRight, 490
AIOEitherToAIONumber, 490
AIOEitherToAIORetType, 490
AIOEitherToDouble, 490

AIOEitherToInt, 490
AIOEitherToShort, 490
AIOEitherToString, 490
AIOEitherToUnsigned, 490
ndkbuild/jni/AIOFifo.c
AIOFifoAllOrNoneInitialize, 493
AIOFifoGetRefSize, 493
AIOFifoGetSize, 493
AIOFifoGetSizeNumElements, 493
AIOFifoInitialize, 493
AIOFifoRead, 493
AIOFifoReadAllOrNone, 493
AIOFifoReadPosition, 493
AIOFifoReadSize, 493
AIOFifoReadSizeNumElements, 493
AIOFifoReset, 493
AIOFifoResize, 493
AIOFifoWrite, 493
AIOFifoWriteAllOrNone, 493
AIOFifoWritePosition, 493
AIOFifoWriteSizeRemaining, 493
AIOFifoWriteSizeRemainingNumElements, 493
DeleteAIOFifo, 493
DeleteAIOFifoTYPE, 493
delta, 493
increment, 493
LOOKUP, 493
NewAIOFifo, 493
NewAIOFifoAllOrNone, 493
NewAIOFifoTYPE, 493
Pop, 493
PopN, 493
Push, 493
PushN, 493
rdelta, 493
ndkbuild/jni/AIOFifo.h
AIOFifo, 498
AIOFifoGetRefSize, 498
AIOFifoGetSize, 499
AIOFifoGetSizeNumElements, 499
AIOFifoRead, 498
AIOFifoReadAllOrNone, 498
AIOFifoReadPosition, 499
AIOFifoReadSize, 499
AIOFifoReadSizeNumElements, 499
AIOFifoReset, 498
AIOFifoResize, 499
AIOFifoTYPE, 498
AIOFifoWrite, 498
AIOFifoWriteAllOrNone, 498
AIOFifoWritePosition, 499
AIOFifoWriteSizeRemaining, 499
AIOFifoWriteSizeRemainingNumElements, 499
DeleteAIOFifo, 498
GRAB_RESOURCE, 497
INPUT_TYPE, 498
NewAIOFifo, 498
NewAIOFifoTYPE, 498
PopN, 499
Push, 498
PushN, 499
RELEASE_RESOURCE, 497
TYPE, 498
ndkbuild/jni/AIOList.c
Deleteint, 500
Deleteintlist, 500
intToString, 500
intlistFirst, 500
intlistInsert, 500
intlistSize, 500
intlistToString, 500
Newintlist, 500
newone, 501
ndkbuild/jni/AIOList.h
Deleteintlist, 503
foreach_CStringArray_p, 503
foreach_int, 503
intlistFirst, 503
intlistInsert, 503
intlistSize, 503
intlistToString, 503
Newintlist, 503
TAIL_Q_LIST, 503
ndkbuild/jni/AIOProductTypes.c
AIOProductGroupContains, 506
AIOProductRangeEnd, 506
AIOProductRangeStart, 506
DeleteAIOProductGroup, 506
DeleteAIOProductRange, 506
groupcpy, 506
NewAIOProductGroup, 506
NewAIOProductRange, 506
ndkbuild/jni/AIOProductTypes.h
AIO_RANGE, 509
AIOProductGroup, 509
AIOProductGroupContains, 509
AIOProductRange, 509
AIOProductRangeEnd, 509
AIOProductRangeStart, 509
DeleteAIOProductGroup, 509
DeleteAIOProductRange, 509
groupcpy, 509
NUMARGS, 509
NewAIOProductGroup, 509
NewAIOProductRange, 509
ndkbuild/jni/AIOTuple.h
AIO_CHAR_ARRAY, 511
AIOTUPLE2, 511
AIOTUPLE2_PTR, 511
AIOTUPLE2_TO_STR, 511
AIOTUPLE2_TYPE, 511
ndkbuild/jni/AIOTypes.h
ACCES_DEPRECATED, 540
ADCalMode, 555
ADGainCode, 552
ADRegister, 551
AIO_ASSERT, 540
AIO_ASSERT_EXIT, 540
AIO_ASSERT_RET, 540
AIO_ASSERT_USB, 541
AIO_ERROR, 541
AIO_MAKE_ERROR, 540
AIO_NUMBER, 542
AIO_SCAN_TYPE, 542
AIOBufferType, 542
AIOCommandCode, 555
AIOContinuousBufMode, 542
AIORESULT, 542
AIORET_TYPE, 542
AIOUSB_BOOL, 542
AIOUSB_BOOL_VAL, 543
AUR_CBUF_EXIT, 540
AUR_CBUF_SETUP, 540
COUNTS, 542
CREATE_ENUM, 539
DACRange, 547
EnumStringLookup, 542
FIFO_Method, 548
FIRST_ENUM, 540
foreach_array, 540
G_BREAKPOINT, 541
G_STMT_END, 541
G_STMT_START, 541
GCC_VERSION, 540

HAS_PTHREAD, 539
LAMBDA, 540
LAST_ENUM, 539
MAX, 540
MAX_VALUE, 540
MIN, 540
MIN_VALUE, 540
NUMBER_CHANNELS, 540
ProductIDS, 543
ROOTCLOCK, 541
ResultCode, 548
THREAD_STATUS, 542
Ushort_Array, 542
VALID_ENUM, 540
VALID_PRODUCT, 540
VENDOR_REQUEST, 552
ndkbuild/jni/AIOUSB_ADC.c
ADC_ADMode, 577
ADC_BulkAcquire, 578
ADC_BulkPoll, 579
ADC_CanCalibrate, 578
ADC_ClearADBuf, 579
ADC_CopyConfig, 575
ADC_CreateADBuf, 579
ADC_GetChannelV, 574
ADC_GetConfig, 575
ADC_GetConfigRegisters, 572
ADC_GetConfigSize, 572
ADC_GetITScanV, 580
ADC_GetMaxClockRate, 577
ADC_GetOversample, 577
ADC_GetScan, 574
ADC_GetScanV, 574
ADC_Initialize, 578
ADC_QueryCal, 578
ADC_Range1, 575
ADC_RangeAll, 575
ADC_ResetDevice, 572
ADC_SetCal, 578
ADC_SetConfig, 575
ADC_SetOversample, 577
ADC_SetScanLimits, 577
ADConfigBlockToYAML, 580
AIOUSB_GetScan, 572
adRanges, 584
adc_get_bulk_data, 572
adcblock_valid_channel_settings, 575
adcblock_valid_size, 575
adcblock_valid_trigger_settings, 575
ConfigureAndBulkAcquire, 581
CreateSmartBuffer, 579
dRef, 584
DoLoadCalTable, 581
GetHiRef, 581
ReadConfigBlock, 572
valid_config_block, 575
WriteConfigBlock, 572
ndkbuild/jni/AIOUSB_ADC.h
ADC_ADMode, 602
ADC_BulkAcquire, 604
ADC_BulkPoll, 604
ADC_CanCalibrate, 603
ADC_GetChannelV, 600
ADC_GetConfig, 602
ADC_GetITScanV, 605
ADC_GetMaxClockRate, 603
ADC_GetOversample, 605
ADC_GetScan, 601
ADC_GetScanV, 600
ADC_Initialize, 603
ADC_QueryCal, 603
ADC_Range1, 602
ADC_RangeAll, 600
ADC_SetCal, 603
ADC_SetConfig, 602
ADC_SetOversample, 605
ADC_SetScanLimits, 603
AIOUSB_GetScan, 601
CreateSmartBuffer, 608
ReadConfigBlock, 605
WriteConfigBlock, 605
ndkbuild/jni/AIOUSB_CTR.c
CTR_8254Load, 626
CTR_8254Mode, 626
CTR_8254ModeLoad, 626
CTR_8254Read, 626
CTR_8254ReadAll, 626
CTR_8254ReadLatched, 627
CTR_8254ReadModeLoad, 626
CTR_8254ReadStatus, 626
CTR_8254SelectGate, 627
CTR_CalculateCountersForClock, 626
CTR_StartOutputFreq, 627
ndkbuild/jni/AIOUSB_CTR.h
CTR_8254Load, 629
CTR_8254Mode, 628
CTR_8254ModeLoad, 629
CTR_8254Read, 629
CTR_8254ReadAll, 629
CTR_8254ReadLatched, 629
CTR_8254ReadModeLoad, 629
CTR_8254ReadStatus, 629
CTR_8254SelectGate, 629
CTR_CalculateCountersForClock, 628
CTR_StartOutputFreq, 629
ndkbuild/jni/AIOUSB_Core.c
AIOUSB_GetCommTimeout, 614
AIOUSB_GetConfigBlock, 614
AIOUSB_GetDevice, 614
AIOUSB_GetMiscClock, 614
AIOUSB_GetVersion, 614
AIOUSB_GetVersionDate, 614
AIOUSB_InitConfigBlock, 614
AIOUSB_Lock, 613
AIOUSB_ResetChip, 613
AIOUSB_SetCommTimeout, 614
AIOUSB_SetMiscClock, 614
AIOUSB_UnLock, 613
AIOUSB_Validate, 613
aio_errno, 615
DeviceTableAtIndex, 613
DeviceTableAtIndex_Lock, 613
GenericVendorRead, 615
GenericVendorWrite, 614
productIDNameTable, 615
ResolveDeviceIndex, 613
ndkbuild/jni/AIOUSB_Core.h
ADC_ResetDevice, 620
AIOError, 620
AIOOption, 620
AIOUSB_GetCommTimeout, 621
AIOUSB_GetConfigBlock, 621
AIOUSB_GetDevice, 621
AIOUSB_GetMiscClock, 621
AIOUSB_GetVersion, 621
AIOUSB_GetVersionDate, 621
AIOUSB_InitConfigBlock, 623
AIOUSB_InitTest, 620
AIOUSB_Lock, 620
AIOUSB_ResetChip, 620
AIOUSB_SetCommTimeout, 621
AIOUSB_SetMiscClock, 621
AIOUSB_UnLock, 620
AIOUSB_Validate, 620

AIOUSB_VoltsToCounts, 621
adRanges, 623
aio_errno, 623
aiousbInit, 623
DeviceTableAtIndex, 620
DeviceTableAtIndex_Lock, 621
GenericVendorRead, 623
GenericVendorWrite, 623
PRIVATE, 620
ndkbuild/jni/AIOUSB_DAC.c
DACDirect, 634
DACMultiDirect, 634
DACOutputClose, 635
DACOutputCloseNoEnd, 635
DACOutputFrame, 635
DACOutputFrameRaw, 635
DACOutputOpen, 635
DACOutputSetCount, 635
DACOutputSetInterlock, 635
DACOutputStart, 635
DACSetBoardRange, 634
ndkbuild/jni/AIOUSB_DAC.h
DACDirect, 637
DACMultiDirect, 637
DACOutputClose, 638
DACOutputCloseNoEnd, 638
DACOutputFrame, 638
DACOutputFrameRaw, 638
DACOutputOpen, 638
DACOutputSetCount, 638
DACOutputSetInterlock, 638
DACOutputStart, 638
DACSetBoardRange, 637
ndkbuild/jni/AIOUSB_DIO.c
aiousb_htons, 640
DIO_ConfigurationQuery, 641
DIO_Configure, 641
DIO_ConfigureEx, 641
DIO_Read1, 641
DIO_Read8, 641
DIO_ReadAll, 641
DIO_StreamClose, 641
DIO_StreamFrame, 641
DIO_StreamOpen, 641
DIO_StreamSetClocks, 641
DIO_Write1, 641
DIO_Write8, 641
DIO_WriteAll, 641
pow_of_minsize, 641
ndkbuild/jni/AIOUSB_DIO.h
DIO_ConfigurationQuery, 645
DIO_Configure, 645
DIO_ConfigureEx, 645
DIO_Read1, 645
DIO_Read8, 645
DIO_ReadAll, 645
DIO_StreamClose, 645
DIO_StreamFrame, 646
DIO_StreamOpen, 645
DIO_StreamSetClocks, 645
DIO_Write1, 645
DIO_Write8, 645
DIO_WriteAll, 645
ndkbuild/jni/AIOUSB_Log.c
cont_thread, 647
message_lock, 647
outfile, 647
ndkbuild/jni/AIOUSB_Log.h
AIOUSB_INFO, 650
AIOUSB_LOG, 650
AIOUSB_WARN, 650
CYAN, 649
cont_thread, 650
GREEN, 649
LOG_LEVEL, 650
MAGENTA, 649
message_lock, 651
outfile, 651
RED, 649
ndkbuild/jni/AIOUSB_Properties.c
AIOUSB_FindDevices, 654
AIOUSB_GetDeviceProperties, 654
AIOUSB_ListDevices, 654
AIOUSB_ShowDevices, 654
GetDeviceBySerialNumber, 653
GetDeviceSerialNumber, 653
non_usb_supported_device, 653
ndkbuild/jni/AIOUSB_Properties.h
AIODisplayType, 656
AIOUSB_FindDevices, 657
AIOUSB_GetDeviceProperties, 657
AIOUSB_ListDevices, 657
AIOUSB_ShowDevices, 657
GetDeviceBySerialNumber, 657
GetDeviceSerialNumber, 657
ndkbuild/jni/AIOUSB_USB.h
usb_bulk_xfer, 659
usb_close, 659
usb_control_xfer, 659
usb_free_devices, 659
usb_get_devices, 659
usb_open, 659
ndkbuild/jni/AIOUSB_WDG.c
DeleteWDGConfig, 661
doSomething, 661
NewWDGConfig, 661
WDG_GetStatus, 662
WDG_Pet, 662
WDG_SetConfig, 661
ndkbuild/jni/AIOUSB_WDG.h
DeleteWDGConfig, 664
NewWDGConfig, 664
WDG_GetStatus, 665
WDG_Pet, 665
WDG_SetConfig, 664
WDGVals, 664
ndkbuild/jni/AIOUSBDevice.c
AIOUSBDeviceCopyADCConfigBlock, 667
AIOUSBDeviceGetADCConfigBlock, 667
AIOUSBDeviceGetDiscardFirstSample, 667
AIOUSBDeviceGetStreamingBlockSize, 667
AIOUSBDeviceGetTesting, 667
AIOUSBDeviceGetTimeout, 667
AIOUSBDeviceGetUSBHandle, 667
AIOUSBDeviceInitializeWithProductID, 667
AIOUSBDeviceSetADCConfigBlock, 667
AIOUSBDeviceSetDiscardFirstSample, 667
AIOUSBDeviceSetTesting, 667
AIOUSBDeviceSetTimeout, 667
AIOUSBDeviceSetUSBHandle, 667
AIOUSBDeviceSize, 667
AIOUSBDeviceToJSON, 667
AIOUSBDeviceWriteADCConfig, 667
DeleteAIOUSBDevice, 667
NewAIOUSBDevice, 667
NewAIOUSBDeviceFromJSON, 667
ndkbuild/jni/AIOUSBDevice.h
AIOUSBDeviceCopyADCConfigBlock, 670
AIOUSBDeviceGetADCConfigBlock, 669
AIOUSBDeviceGetDiscardFirstSample, 670
AIOUSBDeviceGetStreamingBlockSize, 670
AIOUSBDeviceGetTesting, 670
AIOUSBDeviceGetTimeout, 670
AIOUSBDeviceGetUSBHandle, 669

AIOUSBDeviceInitializeWithProductID, 669
AIOUSBDeviceSetADCConfigBlock, 669
AIOUSBDeviceSetDiscardFirstSample, 670
AIOUSBDeviceSetTesting, 670
AIOUSBDeviceSetTimeout, 670
AIOUSBDeviceSetUSBHandle, 669
AIOUSBDeviceSize, 670
AIOUSBDeviceToJSON, 669
AIOUSBDeviceWriteADCConfig, 670
DeviceDescriptor, 669
NewAIOUSBDeviceFromJSON, 669
ndkbuild/jni/CStringArray.c
CStringArrayToString, 680
CStringArrayToStringWithDelimeter, 680
CopyCStringArray, 680
DeleteCStringArray, 680
NewCStringArray, 680
NewCStringArrayFromCArgs, 680
NewCStringArrayWithStrings, 680
ndkbuild/jni/CStringArray.h
CStringArray, 682
CStringArrayToString, 682
CStringArrayToStringWithDelimeter, 682
CopyCStringArray, 682
DeleteCStringArray, 682
NewCStringArray, 682
NewCStringArrayFromCArgs, 682
NewCStringArrayWithStrings, 682
STRING_ARRAY, 682
ndkbuild/jni/DIOBuf.c
DIOBufByteSize, 686
DIOBufGetByteAtIndex, 687
DIOBufGetIndex, 687
DIOBufReplaceBinString, 686
DIOBufReplaceString, 686
DIOBufResize, 686
DIOBufSetByteAtIndex, 687
DIOBufSetIndex, 687
DIOBufSize, 686
DIOBufToBinary, 686
DIOBufToHex, 686
DIOBufToInvertedBinary, 686
DIOBufToString, 686
DeleteDIOBuf, 686
NewDIOBuf, 685
NewDIOBufFromBinStr, 685
NewDIOBufFromChar, 685
ndkbuild/jni/DIOBuf.h
DIOBufByteSize, 692
DIOBufGetByteAtIndex, 693
DIOBufGetIndex, 693
DIOBufReplaceBinString, 692
DIOBufReplaceString, 692
DIOBufResize, 692
DIOBufSetByteAtIndex, 693
DIOBufSetIndex, 693
DIOBufSize, 692
DIOBufToBinary, 692
DIOBufToHex, 692
DIOBufToInvertedBinary, 692
DIOBufToString, 692
DIOBufferType, 691
DeleteDIOBuf, 692
NewDIOBuf, 691
NewDIOBufFromBinStr, 691
NewDIOBufFromChar, 691
ndkbuild/jni/USBDevice.c
AddAllACCESUSBDevices, 748
AddDevice, 748
CopyUSBDevice, 748
DeleteUSBDevice, 749
DeleteUSBDevices, 749
get_usb_device, 749
InitializeUSBDevice, 748
NewUSBDevice, 748
USBDeviceClose, 748
USBDeviceFetchADCConfigBlock, 749
USBDeviceGetIdProduct, 749
USBDeviceGetUSBDeviceHandle, 749
USBDevicePutADCConfigBlock, 749
USBDeviceSetDebug, 749
usb_bulk_transfer, 749
usb_control_transfer, 749
usb_request, 749
usb_reset_device, 749
ndkbuild/jni/USBDevice.h
AddAllACCESUSBDevices, 751
CopyUSBDevice, 751
DeleteUSBDevice, 751
DeleteUSBDevices, 752
get_usb_device, 752
INTERNAL_METHOD, 751
InitializeUSBDevice, 751
LIBUSBArgs, 751
NewUSBDevice, 751
USBDevice, 751
USBDeviceClose, 752
USBDeviceFetchADCConfigBlock, 752
USBDeviceGetIdProduct, 752
USBDeviceGetUSBDeviceHandle, 752
USBDevicePutADCConfigBlock, 752
usb_bulk_transfer, 752
usb_control_transfer, 752
usb_request, 752
usb_reset_device, 752
ndkbuild/jni/cJSON.c
cJSON_AddItemReferenceToArray, 673
cJSON_AddItemReferenceToObject, 673
cJSON_AddItemToArray, 673
cJSON_AddItemToObject, 673
cJSON_AsInteger, 673
cJSON_CreateArray, 673
cJSON_CreateBool, 673
cJSON_CreateDoubleArray, 673
cJSON_CreateFalse, 673
cJSON_CreateFloatArray, 673
cJSON_CreateIntArray, 673
cJSON_CreateNull, 673
cJSON_CreateNumber, 673
cJSON_CreateObject, 673
cJSON_CreateString, 673
cJSON_CreateStringArray, 673
cJSON_CreateTrue, 673
cJSON_Delete, 673
cJSON_DeleteItemFromArray, 673
cJSON_DeleteItemFromObject, 673
cJSON_DetachItemFromArray, 673
cJSON_DetachItemFromObject, 673
cJSON_Duplicate, 673
cJSON_GetArrayItem, 673
cJSON_GetArraySize, 673
cJSON_GetErrorPtr, 673
cJSON_GetObjectItem, 673
cJSON_InitHooks, 673
cJSON_Minify, 673
cJSON_Parse, 673
cJSON_ParseWithOpts, 673
cJSON_Print, 673
cJSON_PrintUnformatted, 673
cJSON_ReplaceItemInArray, 673
cJSON_ReplaceItemInObject, 673
ndkbuild/jni/cJSON.h
cJSON, 678
cJSON_AddBoolToObject, 678

cJSON_AddFalseToObject, 678
cJSON_AddItemReferenceToArray, 679
cJSON_AddItemReferenceToObject, 679
cJSON_AddItemToArray, 679
cJSON_AddItemToObject, 679
cJSON_AddNullToObject, 678
cJSON_AddNumberToObject, 678
cJSON_AddStringToObject, 678
cJSON_AddTrueToObject, 678
cJSON_Array, 678
cJSON_AsInteger, 678
cJSON_CreateArray, 679
cJSON_CreateBool, 678
cJSON_CreateDoubleArray, 679
cJSON_CreateFalse, 678
cJSON_CreateFloatArray, 679
cJSON_CreateIntArray, 679
cJSON_CreateNull, 678
cJSON_CreateNumber, 678
cJSON_CreateObject, 679
cJSON_CreateString, 679
cJSON_CreateStringArray, 679
cJSON_CreateTrue, 678
cJSON_Delete, 678
cJSON_DeleteItemFromArray, 679
cJSON_DeleteItemFromObject, 679
cJSON_DetachItemFromArray, 679
cJSON_DetachItemFromObject, 679
cJSON_Duplicate, 679
cJSON_False, 678
cJSON_GetArrayItem, 678
cJSON_GetArraySize, 678
cJSON_GetErrorPtr, 678
cJSON_GetObjectItem, 678
cJSON_Hooks, 678
cJSON_InitHooks, 678
cJSON_IsReference, 678
cJSON_Minify, 679
cJSON_NULL, 678
cJSON_Number, 678
cJSON_Object, 678
cJSON_Parse, 678
cJSON_ParseWithOpts, 679
cJSON_Print, 678
cJSON_PrintUnformatted, 678
cJSON_ReplaceItemInArray, 679
cJSON_ReplaceItemInObject, 679
cJSON_SetIntValue, 678
cJSON_String, 678
cJSON_True, 678
needs_free
libusb_hotplug_callback, 247
netbsd_backend
libusbi.h, 785
netbsd_usb.c, 795
netbsd_usb.c
netbsd_backend, 795
new_aio_fifo, 260
LOCKING_MECHANISM, 261
Pop, 261
PopN, 261
Push, 261
PushN, 261
NewADCConfigBlockFromJSON
ADCConfigBlock.c, 363
ADCConfigBlock.h, 371
libaiousb.h, 723
ndkbuild/jni/ADCConfigBlock.c, 366
ndkbuild/jni/ADCConfigBlock.h, 374
NewAIOBuf
AIOBuf.c, 375
AIOBuf.h, 377
libaiousb.h, 725
ndkbuild/jni/AIOBuf.c, 376
ndkbuild/jni/AIOBuf.h, 379
NewAIOChannelMask
AIOChannelMask.c, 380
AIOChannelMask.h, 385
libaiousb.h, 723
ndkbuild/jni/AIOChannelMask.c, 383
ndkbuild/jni/AIOChannelMask.h, 388
NewAIOChannelMaskFromChr
AIOChannelMask.c, 382
AIOChannelMask.h, 386
libaiousb.h, 724
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 389
NewAIOChannelMaskFromStr
AIOChannelMask.c, 381
AIOChannelMask.h, 386
libaiousb.h, 723
ndkbuild/jni/AIOChannelMask.c, 384
ndkbuild/jni/AIOChannelMask.h, 388
NewAIOChannelRangeFromStr
AIOChannelRange.c, 391
AIOChannelRange.h, 393
libaiousb.h, 725
ndkbuild/jni/AIOChannelRange.c, 392
ndkbuild/jni/AIOChannelRange.h, 394
NewAIOCmd
AIOCmd.c, 395
AIOCmd.h, 396
ndkbuild/jni/AIOCmd.c, 395
ndkbuild/jni/AIOCmd.h, 396
NewAIOCmdFromJSON
AIOCmd.c, 395
AIOCmd.h, 396
ndkbuild/jni/AIOCmd.c, 395
ndkbuild/jni/AIOCmd.h, 396
NewAIOCommandLineOptionsFromDefaultOptions
AIOCommandLine.c, 398
AIOCommandLine.h, 404
libaiousb.h, 746
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 408
NewAIOConfiguration
AIOConfiguration.c, 410
AIOConfiguration.h, 412
ndkbuild/jni/AIOConfiguration.c, 411
ndkbuild/jni/AIOConfiguration.h, 414
NewAIOContinuousBuf
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 437
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 428
ndkbuild/jni/AIOContinuousBuffer.h, 445
NewAIOContinuousBufForCounts
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 428
ndkbuild/jni/AIOContinuousBuffer.h, 445
NewAIOContinuousBufForVolts
AIOContinuousBuffer.c, 418
AIOContinuousBuffer.h, 438
libaiousb.h, 716
ndkbuild/jni/AIOContinuousBuffer.c, 429
ndkbuild/jni/AIOContinuousBuffer.h, 445
NewAIOContinuousBufFromJSON
AIOContinuousBuffer.c, 425
AIOContinuousBuffer.h, 442
libaiousb.h, 720
ndkbuild/jni/AIOContinuousBuffer.c, 434
ndkbuild/jni/AIOContinuousBuffer.h, 451

NewAIOCountsConverter
AIOCountsConverter.c, 451
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
NewAIOCountsConverterFromAIOContinuousBuf
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.h, 455
NewAIOCountsConverterWithBuffer
AIOCountsConverter.c, 451
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
NewAIOCountsConverterWithScanLimiter
AIOCountsConverter.c, 451
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 455
NewAIODeviceInfo
AIODeviceInfo.c, 456
AIODeviceInfo.h, 457
libaiousb.h, 727
ndkbuild/jni/AIODeviceInfo.c, 457
ndkbuild/jni/AIODeviceInfo.h, 458
NewAIODeviceQuery
AIODeviceQuery.c, 459
AIODeviceQuery.h, 465
libaiousb.h, 727
ndkbuild/jni/AIODeviceQuery.c, 462
ndkbuild/jni/AIODeviceQuery.h, 468
NewAIOFifo
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
NewAIOFifoAllOrNone
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
NewAIOFifoTYPE
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
NewAIOGainRangeFromADCConfigBlock
AIOCountsConverter.c, 452
AIOCountsConverter.h, 454
libaiousb.h, 726
ndkbuild/jni/AIOCountsConverter.c, 453
ndkbuild/jni/AIOCountsConverter.h, 456
NewAIOProductGroup
AIOProductTypes.c, 505
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
NewAIOProductRange
AIOProductTypes.c, 504
AIOProductTypes.h, 507
ndkbuild/jni/AIOProductTypes.c, 506
ndkbuild/jni/AIOProductTypes.h, 509
NewAIOUSBDevice
AIOUSBDevice.c, 666
ndkbuild/jni/AIOUSBDevice.c, 667
NewAIOUSBDeviceFromJSON
AIOUSBDevice.c, 666
AIOUSBDevice.h, 668
libaiousb.h, 743
ndkbuild/jni/AIOUSBDevice.c, 667
ndkbuild/jni/AIOUSBDevice.h, 669
NewCStringArray
CStringArray.c, 679
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
NewCStringArrayFromCArgs
CStringArray.c, 679
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
NewCStringArrayWithStrings
CStringArray.c, 679
CStringArray.h, 681
ndkbuild/jni/CStringArray.c, 680
ndkbuild/jni/CStringArray.h, 682
NewDIOBuf
DIOBuf.c, 682
DIOBuf.h, 688
libaiousb.h, 708
ndkbuild/jni/DIOBuf.c, 685
ndkbuild/jni/DIOBuf.h, 691
NewDIOBufFromBinStr
DIOBuf.c, 683
DIOBuf.h, 688
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 685
ndkbuild/jni/DIOBuf.h, 691
NewDIOBufFromChar
DIOBuf.c, 683
DIOBuf.h, 688
libaiousb.h, 709
ndkbuild/jni/DIOBuf.c, 685
ndkbuild/jni/DIOBuf.h, 691
NewDefaultAIOCommandLineOptions
AIOCommandLine.c, 398
AIOCommandLine.h, 404
libaiousb.h, 746
ndkbuild/jni/AIOCommandLine.c, 400
ndkbuild/jni/AIOCommandLine.h, 408
NewUSBDevice
libaiousb.h, 745
ndkbuild/jni/USBDevice.c, 748
ndkbuild/jni/USBDevice.h, 751
USBDevice.c, 750
USBDevice.h, 753
NewWDGConfig
AIOUSB_WDG.c, 659
AIOUSB_WDG.h, 663
libaiousb.h, 745
ndkbuild/jni/AIOUSB_WDG.c, 661
ndkbuild/jni/AIOUSB_WDG.h, 664
Newintlist
AIOList.c, 499
AIOList.h, 502
ndkbuild/jni/AIOList.c, 500
ndkbuild/jni/AIOList.h, 503
newone
AIOList.c, 500
ndkbuild/jni/AIOList.c, 501
next
aiobuf_iterator, 156
cJSON, 198
list_head, 260
NoDeviceConnected
windows_winusb.h, 829
NoSpeed
windows_usbdk.h, 815
no_argument
getopt.h, 763
node

USB_HUB_NAME_FIXED, 305
node_args, 261
ctx, 261
discdevs, 261
dlink_hdl, 261
last_ugenpath, 261
NodeName
USB_NODE_CONNECTION_NAME, 307
USB_NODE_CONNECTION_NAME_FIXED, 307
NodeType
USB_NODE_INFORMATION, 307
non_usb_supported_device
AIOUSB_Properties.c, 651
ndkbuild/jni/AIOUSB_Properties.c, 653
not_waiting
usbi_cond, 330
null_fd
libusb_testlib_ctx, 254
num_altsetting
libusb_interface, 248
num_channels
aio_counts_converter, 153
AIOCmd, 157
AIOCommandLineOptions, 158
AIOContinuousBuf, 160
opts, 263
num_configurations
libusb_device, 242
num_endpoints
darwin_device_handle_priv::darwin_interface, 215
num_eps
usbfs_streams, 329
num_iso_packets
darwin_transfer_priv, 215
libusb_transfer, 256
usbi_transfer, 333
num_oversamples
aio_counts_converter, 153
AIOCommandLineOptions, 158
AIOContinuousBuf, 160
opts, 263
num_retired
linux_transfer_priv, 259
num_samples
AIOCmd, 157
num_scans
aio_counts_converter, 153
AIOCmd, 157
AIOCommandLineOptions, 158
AIOContinuousBuf, 160
opts, 263, 264
num_streams
usbfs_streams, 329
num_urbs
linux_transfer_priv, 259
numChannels
AIOUSB::AnalogInputSubsystem, 187
AIOUSB::AnalogOutputSubsystem, 192
AIOUSB::DigitalIOSubsystem, 224
numCounterBlocks
AIOUSB::CounterSubsystem, 207
numCounters
AIODeviceQuery, 162
AIOUSB::CounterSubsystem, 207
DeviceInfo, 216
TestCaseSetup, 275
numDIOBytes
AIODeviceQuery, 162
DeviceInfo, 216
TestCaseSetup, 275
numMUXChannels
AIOUSB::AnalogInputSubsystem, 187
numPorts
AIOUSB::DigitalIOSubsystem, 224
numTristateChannels
AIOUSB::DigitalIOSubsystem, 224
numTristateGroups
AIOUSB::DigitalIOSubsystem, 224
number
aio_either_val, 154
number_arguments
AIOArguments, 155
number_channels
config_options, 199
number_of_packets
usbfs_urb, 330
number_oversamples
TestCaseSetup, 275
number_ranges
AIOCommandLineOptions, 158
opts, 264
number_scans
configuration, 200
number_signals
AIOChannelMask, 156
number_to_read
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
NumberFeatureButtonCaps
HIDP_CAPS, 233
NumberFeatureDataIndices
HIDP_CAPS, 233
NumberFeatureValueCaps
HIDP_CAPS, 233
NumberInputButtonCaps
HIDP_CAPS, 233
NumberInputDataIndices
HIDP_CAPS, 233
NumberInputValueCaps
HIDP_CAPS, 233
NumberLinkCollectionNodes
HIDP_CAPS, 233
NumberOfInterfaces
USB_MI_PARENT_INFORMATION, 306
NumberOutputButtonCaps
HIDP_CAPS, 233
NumberOutputDataIndices
HIDP_CAPS, 233
NumberOutputValueCaps
HIDP_CAPS, 233
numports
usbfs_hub_portinfo, 328

OUT
mock_capture_usb.c, 695
mock_dio.c, 697
mock_usb_xfers.c, 698
OPEN_PATTERN
AIOUSB::USBDeviceManager, 325
OS_DARWIN
Xcode/config.h, 755
OS_LINUX
android/config.h, 754
OS_WINDOWS
msvc/config.h, 755
object
aio_either_val, 154
old_action
daitest.cpp, 856
old_stderr
libusb_testlib_ctx, 254
old_stdout
libusb_testlib_ctx, 254
open

AIOUSB::DIOStreamSubsystem, 227
AIOUSB::USBDeviceManager, 323
usbi_os_backend, 332
windows_usb_api_backend, 339
open_count
darwin_cached_device, 214
open_devs
libusb_context, 239
open_devs_lock
libusb_context, 240
openStatus
AIOUSB::USBDeviceManager, 325
openbsd_backend
libusbi.h, 785
openbsd_usb.c, 796
openbsd_usb.c
DEVPATH, 796
openbsd_backend, 796
USBDEV, 796
OperationFailedException, 261
AIOUSB::OperationFailedException, 262
operator<<
AIOUSB, 144
optarg
getopt.c, 762
getopt.h, 763
opterr
AIOCommandLine.c, 399
getopt.c, 762
getopt.h, 763
ndkbuild/jni/AIOCommandLine.c, 402
optind
AIOCommandLine.c, 399
getopt.c, 762
getopt.h, 763
ndkbuild/jni/AIOCommandLine.c, 402
option, 262
flag, 262
has_arg, 262
name, 262
val, 262
optional_argument
getopt.h, 763
options, 262
maxcount, 263
use_maxcount, 263
optopt
getopt.c, 762
getopt.h, 763
opts, 263
adcconfig_json, 264
aiobuf_json, 264
block_size, 264
buffer_size, 264
cal_channel, 264
calibration, 264
calibration_enabled, 264
clock_rate, 264
clock_scale, 264
clock_speed, 264
counts, 264
debug_level, 264
default_aiobuf_json, 264
default_clock_rate, 264
default_end_channel, 264
default_num_channels, 263
default_num_oversamples, 264
default_num_scans, 263
default_start_channel, 264
end_channel, 264
gain_code, 264
index, 264
max_channels, 264
num_channels, 263
num_oversamples, 263
num_scans, 263, 264
number_ranges, 264
outfile, 264
physical, 264
ranges, 264
rate_limit, 264
repeat, 264
reset, 264
slow_acquire, 264
start_channel, 264
verbose, 264
with_timing, 264
orig_usb_bulk_transfer
mock_capture_usb.c, 696
orig_usb_control_transfer
mock_capture_usb.c, 696
orig_usb_get_config
mock_capture_usb.c, 696
orig_usb_put_config
mock_capture_usb.c, 696
orig_usb_request
mock_capture_usb.c, 696
orig_usb_reset_device
mock_capture_usb.c, 696
origconf
USBDevice, 314
original_handle
poll_windows.c, 798
os_count
aio_counts_converter, 153
os_priv
libusb_device, 242
libusb_device_handle, 244
outfile
AIOCommandLineOptions, 158
AIOUSB_Log.c, 646
AIOUSB_Log.h, 649
mock_capture_usb.c, 696
ndkbuild/jni/AIOUSB_Log.c, 647
ndkbuild/jni/AIOUSB_Log.h, 651
opts, 264
output_file
configuration, 200
libusb_testlib_ctx, 254
output_report_size
hid_device_priv, 231
outputMask
DeviceInfo, 216
outputRange
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
OutputReportByteLength
HIDP_CAPS, 233
OutputVoltagePoint, 265
AIOUSB::OutputVoltagePoint, 265
OutputVoltagePointArray, 265
AIOUSB::OutputVoltagePointArray, 266
overSample
AIOUSB::AI16_DataSet, 151
AIOUSB::AnalogInputSubsystem, 187
overlapped
winfd, 340
overlapped_to_winfd
poll_windows.c, 798
poll_windows.h, 800

PERMUTE
getopt.c, 762
PICO_DIO16RO8

AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
pBuf
BulkAcquireWorkerParams, 197
PDEVINST
windows_winusb.h, 826
PDEVNODE
windows_winusb.h, 826
PHIDD_ATTRIBUTES
windows_winusb.h, 829
PHIDP_CAPS
windows_winusb.h, 829
PHIDP_VALUE_CAPS
windows_winusb.h, 829
PID
AIODeviceInfo, 161
AIOUSBDevice, 166
PKLIB_VERSION
windows_winusb.h, 829
POLL_NFDS_TYPE
android/config.h, 754
msvc/config.h, 755
Xcode/config.h, 755
POLLERR
poll_windows.h, 799
POLLHUP
poll_windows.h, 799
POLLIN
poll_windows.h, 799
POLLNVAL
poll_windows.h, 799
POLLOUT
poll_windows.h, 799
POLLPRI
poll_windows.h, 799
PORT_A
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 875
PORT_B
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
PORT_C
dio96_read_write.c, 871
mytest.c, 873
tmp.c, 874
PRIVATE
AIOUSB_Core.h, 616
ndkbuild/jni/AIOUSB_Core.h, 620
PRIX16
inttypes.h, 838
PRIX32
inttypes.h, 838
PRIX64
inttypes.h, 838
PRIX8
inttypes.h, 838
PRIXFAST16
inttypes.h, 839
PRIXFAST32
inttypes.h, 839
PRIXFAST64
inttypes.h, 839
PRIXFAST8
inttypes.h, 839
PRIXLEAST16
inttypes.h, 839
PRIXLEAST32
inttypes.h, 839
PRIXLEAST64
inttypes.h, 839
PRIXLEAST8
inttypes.h, 838
PRIXMAX
inttypes.h, 839
PRIXPTR
inttypes.h, 840
PRId16
inttypes.h, 837
PRId32
inttypes.h, 837
PRId64
inttypes.h, 837
PRId8
inttypes.h, 837
PRIdFAST16
inttypes.h, 837
PRIdFAST32
inttypes.h, 837
PRIdFAST64
inttypes.h, 837
PRIdFAST8
inttypes.h, 837
PRIdLEAST16
inttypes.h, 837
PRIdLEAST32
inttypes.h, 837
PRIdLEAST64
inttypes.h, 837
PRIdLEAST8
inttypes.h, 837
PRIdMAX
inttypes.h, 837
PRIdPTR
inttypes.h, 840
PRIi16
inttypes.h, 837
PRIi32
inttypes.h, 837
PRIi64
inttypes.h, 837
PRIi8
inttypes.h, 837
PRIiFAST16
inttypes.h, 837
PRIiFAST32
inttypes.h, 837
PRIiFAST64
inttypes.h, 837
PRIiFAST8
inttypes.h, 837
PRIiLEAST16
inttypes.h, 837
PRIiLEAST32
inttypes.h, 837
PRIiLEAST64
inttypes.h, 837
PRIiLEAST8
inttypes.h, 837
PRIiMAX
inttypes.h, 837
PRIiPTR
inttypes.h, 840
PRIo16
inttypes.h, 837
PRIo32
inttypes.h, 837
PRIo64
inttypes.h, 837
PRIo8
inttypes.h, 837
PRIoFAST16
inttypes.h, 838

PRIoFAST32
inttypes.h, 838
PRIoFAST64
inttypes.h, 838
PRIoFAST8
inttypes.h, 837
PRIoLEAST16
inttypes.h, 837
PRIoLEAST32
inttypes.h, 837
PRIoLEAST64
inttypes.h, 837
PRIoLEAST8
inttypes.h, 837
PRIoMAX
inttypes.h, 838
PRIoPTR
inttypes.h, 840
PRIu16
inttypes.h, 838
PRIu32
inttypes.h, 838
PRIu64
inttypes.h, 838
PRIu8
inttypes.h, 838
PRIuFAST16
inttypes.h, 838
PRIuFAST32
inttypes.h, 838
PRIuFAST64
inttypes.h, 838
PRIuFAST8
inttypes.h, 838
PRIuLEAST16
inttypes.h, 838
PRIuLEAST32
inttypes.h, 838
PRIuLEAST64
inttypes.h, 838
PRIuLEAST8
inttypes.h, 838
PRIuMAX
inttypes.h, 838
PRIuPTR
inttypes.h, 840
PRIx16
inttypes.h, 838
PRIx32
inttypes.h, 838
PRIx64
inttypes.h, 838
PRIx8
inttypes.h, 838
PRIxFAST16
inttypes.h, 838
PRIxFAST32
inttypes.h, 838
PRIxFAST64
inttypes.h, 838
PRIxFAST8
inttypes.h, 838
PRIxLEAST16
inttypes.h, 838
PRIxLEAST32
inttypes.h, 838
PRIxLEAST64
inttypes.h, 838
PRIxLEAST8
inttypes.h, 838
PRIxMAX
inttypes.h, 838
PRIxPTR
inttypes.h, 840
PROD_NAME_SIZE
AIOUSB_Core.h, 616
ndkbuild/jni/AIOUSB_Core.h, 620
PTRDIFF_MAX
stdint.h, 844
PTRDIFF_MIN
stdint.h, 844
PUBLIC_EXTERN
AIOUSB_Core.h, 616
ndkbuild/jni/AIOUSB_Core.h, 620
PUKW_DEVICE
wince_usb.h, 808
PUSB_DK_DEVICE_ID
windows_usbdk.h, 814
PUSB_PIPE_INFO
windows_winusb.h, 827
PUSB_PROTOCOLS
windows_winusb.h, 827
packet_count
usb_raw_command, 311
packet_descriptors
usb_raw_command, 311
parent
AIOUSB::Counter, 203
AIOUSB::DeviceSubsystem, 219
parent_dev
libusb_device, 242
windows_device_priv, 338
parent_session
darwin_cached_device, 213
pass_through
AIOCommandLineOptions, 158
path
windows_device_priv, 338
PendingDACData
AIOUSBDevice, 167
perftest.c
main, 879
phypath
sunos_device_priv, 269
physical
AIOCommandLineOptions, 159
opts, 264
pid
fx_known_device, 231
hid_device_priv, 231
pipe_number
poll_windows.c, 798
PipeId
WINUSB_PIPE_INFORMATION, 342
PipeType
WINUSB_PIPE_INFORMATION, 342
points
AIOUSB::AI16_DataSet, 151
poll_windows.h
RW_NONE, 800
RW_READ, 800
RW_WRITE, 800
WINDOWS_2003, 800
WINDOWS_7, 800
WINDOWS_8, 800
WINDOWS_8_1_OR_LATER, 800
WINDOWS_CE, 800
WINDOWS_MAX, 800
WINDOWS_UNDEFINED, 800
WINDOWS_UNSUPPORTED, 800
WINDOWS_VISTA, 800
WINDOWS_XP, 800
poll_dbg
poll_windows.c, 797

poll_fd
poll_windows.c, 798
poll_posix.c
usbi_pipe, 796
poll_posix.h
usbi_close, 796
usbi_pipe, 797
usbi_poll, 796
usbi_read, 796
usbi_write, 796
poll_windows.c
exit_polling, 797
fd_to_winfd, 797
handle_to_winfd, 798
INVALID_WINFD, 798
init_polling, 797
is_polling_set, 798
LPOVERLAPPED, 798
mutex, 798
original_handle, 798
overlapped_to_winfd, 798
pipe_number, 798
poll_dbg, 797
poll_fd, 798
thread_id, 798
usbi_close, 798
usbi_create_fd, 797
usbi_free_fd, 797
usbi_pipe, 797
usbi_poll, 798
usbi_read, 798
usbi_write, 798
Use_Duplicate_Handles, 797
poll_windows.h
cancel_transfer, 800
DUMMY_HANDLE, 799
exit_polling, 800
fd_to_winfd, 800
handle_to_winfd, 800
HasOverlappedIoCompletedSync, 799
INVALID_WINFD, 800
init_polling, 800
MAX_FDS, 799
overlapped_to_winfd, 800
POLLERR, 799
POLLHUP, 799
POLLIN, 799
POLLNVAL, 799
POLLOUT, 799
POLLPRI, 799
rw_type, 800
STATUS_REPARSE, 799
timersub, 799
usbi_close, 800
usbi_create_fd, 800
usbi_free_fd, 800
usbi_pipe, 800
usbi_poll, 800
usbi_read, 800
usbi_write, 800
windows_version, 800
pollable_fd
wince_transfer_priv, 337
windows_transfer_priv, 339
pollfd, 266
events, 266
fd, 266
revents, 266
usbi_pollfd, 333
pollfds
libusb_context, 240
pollfds_cnt
libusb_context, 240
Polling and timing, 131
libusb_event_handler_active, 135
libusb_event_handling_ok, 135
libusb_free_pollfds, 139
libusb_get_next_timeout, 138
libusb_get_pollfds, 139
libusb_handle_events, 137
libusb_handle_events_completed, 137
libusb_handle_events_locked, 137
libusb_handle_events_timeout, 136
libusb_handle_events_timeout_completed, 136
libusb_interrupt_event_handler, 135
libusb_lock_event_waiters, 135
libusb_lock_events, 134
libusb_pollfd_added_cb, 133
libusb_pollfd_removed_cb, 134
libusb_pollfds_handle_timeouts, 137
libusb_set_pollfd_notifiers, 138
libusb_try_lock_events, 134
libusb_unlock_event_waiters, 136
libusb_unlock_events, 134
libusb_wait_for_event, 136
Pop
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
new_aio_fifo, 261
PopN
AIOContinuousBuf, 161
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
new_aio_fifo, 261
Port
tag_USB_DK_DEVICE_INFO, 270
port
darwin_cached_device, 214
usbfs_hub_portinfo, 328
windows_device_priv, 338
port_number
libusb_device, 242
pos
AIOChannelMask, 156
pow_of_minsize
AIOUSB_DIO.c, 639
ndkbuild/jni/AIOUSB_DIO.c, 641
prev
cJSON, 198
list_head, 260
print
AIOUSB::AI16_DataSet, 151
AIOUSB::AnalogInputSubsystem, 172
AIOUSB::AnalogOutputSubsystem, 191
AIOUSB::CounterSubsystem, 205
AIOUSB::DeviceSubsystem, 218
AIOUSB::DigitalIOSubsystem, 220
AIOUSB::DIOStreamSubsystem, 226
AIOUSB::USB_AI16_Family, 279
AIOUSB::USB_AIO16_Family, 282
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_CTR_15_Family, 288
AIOUSB::USB_DA12_8A_Family, 291
AIOUSB::USB_DA12_8E_Family, 294
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 299
AIOUSB::USB_DIO_Family, 301
AIOUSB::USBDeviceBase, 316
AIOUSB::USBDeviceManager, 321
print_aio_usage
aiocommon.c, 849

aiocommon.h, 850
print_usage
bulk_acquire_sample.c, 853
slow_receiver_test.cpp, 859
printDevices
AIOUSB::USBDeviceManager, 321
process_aio_cmd_line
aiocommon.c, 849
aiocommon.h, 850
process_cmd_line
bulk_acquire_sample.c, 853
continuous_mode_callback.c, 855
continuous_mode_from_json_config.c, 855
dirktest.c, 857
simple_continuous_with_json.c, 864
slow_receiver_test.cpp, 859
start_stop_continuous.c, 865
test.c, 860
ProductIDS_begin
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
ProductIDS_end
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
product_id
libusb_hotplug_callback, 247
ProductID
AIOUSBDevice, 166
DeviceProperties, 217
HIDD_ATTRIBUTES, 232
productID
AIODeviceQuery, 162
AIOUSB::USBDeviceBase, 318
DeviceInfo, 216
TestCaseSetup, 275
ProductIDName, 266
id, 266
name, 266
productIDNameTable
AIOUSB_Core.c, 612
ndkbuild/jni/AIOUSB_Core.c, 615
ProductIDS
AIOTypes.h, 521
ndkbuild/jni/AIOTypes.h, 543
ProductIDToName
AIODeviceTable.c, 471
AIODeviceTable.h, 479
libaiousb.h, 730
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIODeviceTable.h, 482
productIDToName
AIOUSB::USBDeviceManager, 321, 322
ProductNameToID
AIODeviceTable.c, 471
AIODeviceTable.h, 479
libaiousb.h, 730
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIODeviceTable.h, 482
productNameToID
AIOUSB::USBDeviceManager, 322, 323
ptrdiff_t
stdint.h, 845
Push
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 498
new_aio_fifo, 261
PushN
AIOContinuousBuf, 161
AIOFifo.c, 491
AIOFifo.h, 496
libaiousb.h, 731
ndkbuild/jni/AIOFifo.c, 493
ndkbuild/jni/AIOFifo.h, 499
new_aio_fifo, 261

QueryDeviceInfo
AIODeviceTable.c, 470
AIODeviceTable.h, 478
libaiousb.h, 729
ndkbuild/jni/AIODeviceTable.c, 474
ndkbuild/jni/AIODeviceTable.h, 482
QueryDeviceInformation
winusb_interface, 341
QueryInterfaceSettings
winusb_interface, 341
QueryPipe
winusb_interface, 341
quot
imaxdiv_t, 234

REQUIRE_ORDER
getopt.c, 762
RETURN_IN_ORDER
getopt.c, 762
RUNNING
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
RUNNING_OR_WITH_DATA
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
RW_NONE
poll_windows.h, 800
RW_READ
poll_windows.h, 800
RW_WRITE
poll_windows.h, 800
RANGE_0_10V
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
RANGE_0_1V
AIOUSB::AnalogInputSubsystem, 186
RANGE_0_2_5V
AIOUSB::DA12_AnalogOutputSubsystem, 211
RANGE_0_2V
AIOUSB::AnalogInputSubsystem, 186
RANGE_0_5V
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 211
RANGE_10V
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
RANGE_1V
AIOUSB::AnalogInputSubsystem, 186
RANGE_2_5V
AIOUSB::DA12_AnalogOutputSubsystem, 211
RANGE_2V
AIOUSB::AnalogInputSubsystem, 186
RANGE_5V
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 211
RANGE_TEXT
AIOUSB::AnalogInputSubsystem, 186
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DA12_AnalogOutputSubsystem, 212
RATE_LIMIT
idio_sample.c, 878
idio_sample2.c, 878
iiro_sample.c, 880

RAW_IO
windows_winusb.h, 826
READ
sunos_usb.h, 802
READ_CAPACITY_LENGTH
xusb.c, 766
RED
AIOUSB_Log.h, 647
ndkbuild/jni/AIOUSB_Log.h, 649
REG_CAL_MODE
AIOUSB::AnalogInputSubsystem, 187
REG_GAIN_CODE
AIOUSB::AnalogInputSubsystem, 187
REG_OVERSAMPLE
AIOUSB::AnalogInputSubsystem, 187
REG_START_END
AIOUSB::AnalogInputSubsystem, 187
REG_TRIG_MODE
AIOUSB::AnalogInputSubsystem, 187
RELEASE_RESOURCE
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 497
REPEAT
AIOCommandLine.h, 404
aiocommon.c, 849
ndkbuild/jni/AIOCommandLine.h, 407
REQUEST_SENSE_LENGTH
xusb.c, 766
RETRY_LIMIT
ezusb.c, 759
RETRY_MAX
xusb.c, 766
RETURN_TYPE
windows_winusb.h, 826
ROOTCLOCK
AIOContinuousBuffer.h, 437
AIOTypes.h, 519
ndkbuild/jni/AIOContinuousBuffer.h, 445
ndkbuild/jni/AIOTypes.h, 541
RW_INTERNAL
ezusb.c, 759
RW_MEMORY
ezusb.c, 759
ram_mode
ezusb.c, 759
ram_poke_context, 266
count, 267
device, 267
mode, 267
total, 267
range
ADRange, 146
AIOUSB::AI16_DataPoint, 148
AIOUSB::AnalogIORange, 190
rangeCounts
AIOUSB::AnalogIORange, 190
RangeShift
AIOUSBDevice, 166
RangeValueLookup
AIOContinuousBuffer.c, 417
ndkbuild/jni/AIOContinuousBuffer.c, 428
rangeVolts
AIOUSB::AnalogIORange, 190
rangelookup, 267
maxvalue, 267
minvalue, 267
ranges
AIOCommandLineOptions, 159
opts, 264
rate_limit
AIOCommandLineOptions, 159
opts, 264
raw_cfgdescr
sunos_device_priv, 269
RawCountsWorkFunction
AIOContinuousBuffer.c, 417
ndkbuild/jni/AIOContinuousBuffer.c, 428
rc
libusb_version, 258
rdelta
AIOFifo.c, 491
ndkbuild/jni/AIOFifo.c, 493
read
AIOUSB::AnalogInputSubsystem, 181
AIOUSB::DigitalIOSubsystem, 222, 223
AIOUSB::DIOStreamSubsystem, 228
read_and_write.c
CHECK_RESULT, 871
find_dio, 871
main, 871
read_channels_test.c
find_ai_board, 862
main, 862
read_channels_with_getchannelv_test.cpp
get_options, 862
main, 862
readBuffer
DeviceInfo, 216
readBulkBuffer
AIOUSB::AnalogInputSubsystem, 187
readBulkNext
AIOUSB::AnalogInputSubsystem, 183
readBulkSamplesAvailable
AIOUSB::AnalogInputSubsystem, 183
readBulkSamplesRequested
AIOUSB::AnalogInputSubsystem, 187
readBulkSamplesRetrieved
AIOUSB::AnalogInputSubsystem, 187
readBulkStart
AIOUSB::AnalogInputSubsystem, 182
readConfig
AIOUSB::AnalogInputSubsystem, 174
ReadConfigBlock
AIOUSB_ADC.c, 559
AIOUSB_ADC.h, 592
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 572
ndkbuild/jni/AIOUSB_ADC.h, 605
readCount
AIOUSB::Counter, 202
readCountAndSetModeAndCount
AIOUSB::Counter, 203
readCountAndStatus
AIOUSB::Counter, 202
readCounts
AIOUSB::AnalogInputSubsystem, 182
AIOUSB::CounterSubsystem, 205
ReadPipe
winusb_interface, 341
readVolts
AIOUSB::AnalogInputSubsystem, 182
reap_action
linux_transfer_priv, 259
linux_usbfs.c, 791
reap_status
linux_transfer_priv, 259
receiver.cpp
main, 867
refcnt
libusb_device, 242
refcount
darwin_cached_device, 214
reg_prop
driver_lookup, 229

registers
ADCConfigBlock, 146
release_interface
usbi_os_backend, 332
windows_usb_api_backend, 339
ReleaseInterface
USBDevice, 313
USBDeviceHandle, 319
rem
imaxdiv_t, 234
repeat
opts, 264
repeat_number
AIOCommandLineOptions, 159
req
darwin_transfer_priv, 215
USB_CONFIGURATION_DESCRIPTOR_SHORT, 287
request
usb_raw_command, 310
WINUSB_SETUP_PACKET, 342
request_type
usb_raw_command, 310
WINUSB_SETUP_PACKET, 342
required_argument
getopt.h, 763
Reserved
HIDP_CAPS, 233
ReservedMBZ
USB_HUB_CAP_FLAGS, 303
reset
AIOCommandLineOptions, 158
AIOUSB::USBDeviceBase, 317
opts, 264
reset_device
usbi_os_backend, 332
windows_usb_api_backend, 340
resetCPU
TestCaseSetup, 275
ResetCounters
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
ResetDevice
winusb_interface, 341
ResetPipe
winusb_interface, 341
ResolveDeviceIndex
AIOUSB_Core.c, 610
ndkbuild/jni/AIOUSB_Core.c, 613
restricted_functionality
windows_device_priv, 338
Result
tag_USB_DK_TRANSFER_REQUEST, 271
result
darwin_transfer_priv, 215
ResultCode_begin
AIOTypes.h, 527, 528
ndkbuild/jni/AIOTypes.h, 549, 550
ResultCode_end
AIOTypes.h, 528, 529
ndkbuild/jni/AIOTypes.h, 550, 551
ResultCode
AIOTypes.h, 526
ndkbuild/jni/AIOTypes.h, 548
retval
aioerror, 162
revents
pollfd, 266
reverse_cal_table.cpp
goDoIt, 859
main, 859
right
aio_ret_value, 154
root
USB_HUB_NAME_FIXED, 305
RootClock
AIOUSBDevice, 166
DeviceProperties, 217
RootHubName
USB_ROOT_HUB_NAME, 311
USB_ROOT_HUB_NAME_FIXED, 311
RosterLooper, 267
fInitCheck, 268
fMessenger, 268
fRoot, 267
fRoster, 267
InitCheck, 267
MessageReceived, 267
RosterLooper, 267
RosterLooper, 267
Stop, 267
run
configuration, 200
run_acquisition
start_stop_continuous.c, 865
rw
winfd, 340
rw_type
poll_windows.h, 800

SAMPLE_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
SETCAL_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
START_CHANNEL
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
STATE_AWAIT_IMAGE
dpfp.c, 757
dpfp_threaded.c, 758
STATE_AWAIT_IRQ_FINGER_DETECTED
dpfp.c, 757
dpfp_threaded.c, 758
STATE_AWAIT_IRQ_FINGER_REMOVED
dpfp.c, 757
dpfp_threaded.c, 758
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_OFF
dpfp.c, 757
dpfp_threaded.c, 758
STATE_AWAIT_MODE_CHANGE_AWAIT_FINGER_ON
dpfp.c, 757
dpfp_threaded.c, 758
STATE_AWAIT_MODE_CHANGE_CAPTURE
dpfp.c, 757
dpfp_threaded.c, 758
SUBMIT_FAILED
linux_usbfs.c, 791
SUCCESS
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 877
SCNd16
inttypes.h, 839
SCNd32
inttypes.h, 839
SCNd64
inttypes.h, 839
SCNdFAST16
inttypes.h, 839
SCNdFAST32
inttypes.h, 839
SCNdFAST64

inttypes.h, 839
SCNdLEAST16
inttypes.h, 839
SCNdLEAST32
inttypes.h, 839
SCNdLEAST64
inttypes.h, 839
SCNdMAX
inttypes.h, 839
SCNdPTR
inttypes.h, 840
SCNi16
inttypes.h, 839
SCNi32
inttypes.h, 839
SCNi64
inttypes.h, 839
SCNiFAST16
inttypes.h, 839
SCNiFAST32
inttypes.h, 839
SCNiFAST64
inttypes.h, 839
SCNiLEAST16
inttypes.h, 839
SCNiLEAST32
inttypes.h, 839
SCNiLEAST64
inttypes.h, 839
SCNiMAX
inttypes.h, 839
SCNiPTR
inttypes.h, 840
SCNo16
inttypes.h, 839
SCNo32
inttypes.h, 839
SCNo64
inttypes.h, 839
SCNoFAST16
inttypes.h, 839
SCNoFAST32
inttypes.h, 840
SCNoFAST64
inttypes.h, 840
SCNoLEAST16
inttypes.h, 839
SCNoLEAST32
inttypes.h, 839
SCNoLEAST64
inttypes.h, 839
SCNoMAX
inttypes.h, 840
SCNoPTR
inttypes.h, 840
SCNu16
inttypes.h, 840
SCNu32
inttypes.h, 840
SCNu64
inttypes.h, 840
SCNuFAST16
inttypes.h, 840
SCNuFAST32
inttypes.h, 840
SCNuFAST64
inttypes.h, 840
SCNuLEAST16
inttypes.h, 840
SCNuLEAST32
inttypes.h, 840
SCNuLEAST64
inttypes.h, 840
SCNuMAX
inttypes.h, 840
SCNuPTR
inttypes.h, 840
SCNx16
inttypes.h, 840
SCNx32
inttypes.h, 840
SCNx64
inttypes.h, 840
SCNxFAST16
inttypes.h, 840
SCNxFAST32
inttypes.h, 840
SCNxFAST64
inttypes.h, 840
SCNxLEAST16
inttypes.h, 840
SCNxLEAST32
inttypes.h, 840
SCNxLEAST64
inttypes.h, 840
SCNxMAX
inttypes.h, 840
SCNxPTR
inttypes.h, 840
SEM_NAME
dpfp_threaded.c, 758
SIG_ATOMIC_MAX
stdint.h, 844
SIG_ATOMIC_MIN
stdint.h, 844
SIZE_MAX
stdint.h, 844
SPDRP_ADDRESS
windows_winusb.h, 823
STATE
AIOChannelRange.c, 391
ndkbuild/jni/AIOChannelRange.c, 392
STATUS_HALT_FLAG
wince_usb.h, 808
STATUS_REPARSE
poll_windows.h, 799
STRING_ARRAY
CStringArray.h, 681
ndkbuild/jni/CStringArray.h, 682
SUB_API_LIBUSB0
windows_winusb.h, 824
SUB_API_LIBUSBK
windows_winusb.h, 824
SUB_API_MAX
windows_winusb.h, 824
SUB_API_NOTSET
windows_winusb.h, 824
SUB_API_WINUSB
windows_winusb.h, 824
SUSPEND_DELAY
windows_winusb.h, 826
SWAP_FLAGS
getopt.c, 762
SYSFS_DEVICE_PATH
linux_usbfs.h, 793
safe_closehandle
windows_common.h, 809
safe_free
windows_common.h, 809
safe_min
windows_common.h, 809
safe_sprintf
windows_common.h, 810
safe_stprintf

windows_common.h, 810
safe_strcat
windows_common.h, 810
safe_strcmp
windows_common.h, 810
safe_strcp
windows_common.h, 809
safe_strcpy
windows_common.h, 809
safe_stricmp
windows_common.h, 810
safe_strlen
windows_common.h, 810
safe_strncat
windows_common.h, 810
safe_strncmp
windows_common.h, 810
safe_tcslen
windows_common.h, 810
safe_unref_device
windows_common.h, 810
sam3u_benchmark.c
EP_DATA_IN, 765
EP_ISO_IN, 765
main, 765
sample3.c
main, 870
sample_dio.c
find_ai_board, 859
main, 859
SampleClass.cpp
main, 866
samples/TestLib/TestCaseSetup.cpp, 850
samples/TestLib/TestCaseSetup.h, 851
samples/TestLib/aiocommon.c, 849
samples/TestLib/aiocommon.h, 850
samples/USB-AI16-16/HOLD/dirktest.c, 857
samples/USB-AI16-16/HOLD/julian_test.c, 858
samples/USB-AI16-16/HOLD/reverse_cal_table.cpp, 858
samples/USB-AI16-16/HOLD/sample_dio.c, 859
samples/USB-AI16-16/HOLD/slow_receiver_test.cpp, 859
samples/USB-AI16-16/HOLD/test.c, 860
samples/USB-AI16-16/README.doc, 352
samples/USB-AI16-16/android/README.md, 880
samples/USB-AI16-16/android/read_channels_test/README.md, 880
samples/USB-AI16-16/bulk_acquire_sample.c, 852
samples/USB-AI16-16/burst_test.c, 853
samples/USB-AI16-16/continuous_mode.c, 854
samples/USB-AI16-16/continuous_mode_callback.c, 854
samples/USB-AI16-16/continuous_mode_from_json_config.c, 855
samples/USB-AI16-16/daitest.cpp, 856
samples/USB-AI16-16/dio_sample.c, 856
samples/USB-AI16-16/diotest.c, 857
samples/USB-AI16-16/diotest2.cpp, 857
samples/USB-AI16-16/java/README.md, 880
samples/USB-AI16-16/java/extcal/README.md, 880
samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp, 860
samples/USB-AI16-16/java/read_channels_test/README.md, 880
samples/USB-AI16-16/java/read_channels_test/nativeutils/README.md, 880
samples/USB-AI16-16/jni/read_channels_test.c, 861
samples/USB-AI16-16/read_channels_test.c, 862
samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp, 862
samples/USB-AI16-16/reverse_cal_table.cpp, 858
samples/USB-AI16-16/sample.cpp, 863
samples/USB-AI16-16/simp_test.cpp, 864
samples/USB-AI16-16/simple_continuous_with_json.c, 864
samples/USB-AI16-16/start_stop_continuous.c, 865
samples/USB-AI16-16/test_fastscan.cpp, 865
samples/USB-AO16-16/README.doc, 352
samples/USB-AO16-16/sample.cpp, 863
samples/USB-ARB1/stream_test.c, 866
samples/USB-DA12-8A/README.doc, 352
samples/USB-DA12-8A/SampleClass.cpp, 866
samples/USB-DA12-8A/sample.cpp, 863
samples/USB-DIO-16/README.doc, 352
samples/USB-DIO-16/receiver.cpp, 866
samples/USB-DIO-16/sample.cpp, 863
samples/USB-DIO-16/standalone_receiver.c, 867
samples/USB-DIO-32/README.doc, 352
samples/USB-DIO-32/daisample.c, 867
samples/USB-DIO-32/read_and_write_sample.c, 867
samples/USB-DIO-32/sample3.c, 869
samples/USB-DIO-48/read_and_write.c, 870
samples/USB-DIO-96/dio96_read_write.c, 871
samples/USB-DIO-96/mytest.c, 872
samples/USB-DIO-96/read_and_write_sample.c, 868
samples/USB-DIO-96/tmp.c, 874
samples/USB-DIO-96/write_sample.c, 875
samples/USB-IDIO-16_8/README.doc, 352
samples/USB-IDIO-16_8/idio_sample.c, 877
samples/USB-IDIO-16_8/idio_sample2.c, 878
samples/USB-IDIO-16_8/perftest.c, 879
samples/USB-IIRO-16_8/README.doc, 352
samples/USB-IIRO-16_8/iiro_sample.c, 879
save_results
mock_dio.c, 697
scan_count
aio_counts_converter, 153
scan_type
configuration, 200
scanForDevices
AIOUSB::USBDeviceManager, 324
scans_read
AIOContinuousBuf, 160
ScheduleOffset
USB_PIPE_INFO, 308
selectGate
AIOUSB::CounterSubsystem, 206
SerialNumber
DeviceProperties, 217
serialNumber
AIOUSB::USBDeviceBase, 318
DeviceInfo, 216
session
darwin_cached_device, 213
session_data
libusb_device, 242
set_configuration
usbi_os_backend, 332
set_interface_altsetting
usbi_os_backend, 332
windows_usb_api_backend, 339
SetActiveConfiguration
USBDevice, 313
SetAltSetting
USBDeviceHandle, 319
setAutoConfig
AIOUSB::AnalogInputSubsystem, 173
setCalMode
AIOUSB::AnalogInputSubsystem, 175
setCalibrationTable
AIOUSB::AnalogInputSubsystem, 179
SetCancelled
USBTransfer, 335
setClock
AIOUSB::AnalogInputSubsystem, 180

AIOUSB::DIOStreamSubsystem, 227
setCommTimeout
AIOUSB::USBDeviceBase, 316
SetConfig
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
SetConfiguration
USBDeviceHandle, 319
setCount
AIOUSB::Counter, 202
setCountRange
AIOUSB::AnalogIORange, 189
SetCurrentAlternateSetting
winusb_interface, 341
setCurrentDeviceIndex
TestCaseSetup, 273
setDifferentialMode
AIOUSB::AnalogInputSubsystem, 177, 178
setDiscardFirstSample
AIOUSB::AnalogInputSubsystem, 174
setMaxCount
TestCaseSetup, 274
setMiscClock
AIOUSB::USBDeviceBase, 315
setMode
AIOUSB::Counter, 201
setModeAndCount
AIOUSB::Counter, 202
setOverSample
AIOUSB::AnalogInputSubsystem, 179
SetPipePolicy
winusb_interface, 341
SetPowerPolicy
winusb_interface, 341
setRange
AIOUSB::AI16_InputRange, 152
AIOUSB::AnalogInputSubsystem, 176
AIOUSB::AnalogIORange, 189
AIOUSB::AO16_AnalogOutputSubsystem, 194
AIOUSB::AO16_OutputRange, 196
AIOUSB::DA12_AnalogOutputSubsystem, 209, 210
AIOUSB::DA12_OutputRange, 213
setRangeAndDiffMode
AIOUSB::AnalogInputSubsystem, 178, 179
setScanRange
AIOUSB::AnalogInputSubsystem, 172
setStreamingBlockSize
AIOUSB::AnalogInputSubsystem, 180
AIOUSB::DIOStreamSubsystem, 226
AIOUSB::USBDeviceBase, 316
setTriggerMode
AIOUSB::AnalogInputSubsystem, 175
setVoltRange
AIOUSB::AnalogIORange, 189
SetupPacket
USB_CONFIGURATION_DESCRIPTOR_SHORT, 287
USB_DESCRIPTOR_REQUEST, 295
setupVoltageParameters
TestCaseSetup, 275
show_byte
dio96_read_write.c, 872
mytest.c, 873
tmp.c, 875
signal_index
AIOChannelMask, 156
signal_indices
AIOChannelMask, 156
signals
AIOChannelMask, 156
signr
usbfs_urb, 330
simp_test.cpp
main, 864
simple_continuous_with_json.c
fnd, 865
fp, 865
get_channel_range, 864
main, 865
process_cmd_line, 864
Size
HIDD_ATTRIBUTES, 232
size
ADCConfigBlock, 145
aio_ret_value, 154
AIOArgument, 155
AIOBuf, 155
AIOChannelMask, 156
AIOContinuousBuf, 160
darwin_transfer_priv, 215
DIOBuf, 225
ushort_array, 335
skip_external
ezusb.c, 759
skip_internal
ezusb.c, 759
SleepEx
missing.h, 841
slow
usbfs_connectinfo, 326
slow_acquire
AIOCommandLineOptions, 159
opts, 264
slow_receiver_test.cpp
main, 859
print_usage, 859
process_cmd_line, 859
Speed
tag_USB_DK_DEVICE_INFO, 270
USB_NODE_CONNECTION_INFORMATION_EX, 306
speed
libusb_device, 242
standalone_receiver.c
main, 867
Start
USBRoster, 334
start
aio_channel_range, 153
start_channel
AIOChannelRangeTmp, 157
AIOCommandLineOptions, 158
channel_range, 197
opts, 264
start_frame
usbfs_urb, 330
start_scanning
AIOContinuousBuf, 160
start_stop_continuous.c
fnd, 865
fp, 865
get_channel_range, 865
main, 865
process_cmd_line, 865
run_acquisition, 865
startChannel
AIOUSB::AnalogInputSubsystem, 187
startClock
AIOUSB::CounterSubsystem, 206
StartStreaming
AIOContinuousBuffer.c, 422
ndkbuild/jni/AIOContinuousBuffer.c, 432
state_flags
usbi_transfer, 333

statfd
endpoint, 230
status
AIOContinuousBuf, 161
libusb_iso_packet_descriptor, 251
libusb_transfer, 256
usb_raw_command, 310
usbfs_iso_packet_desc, 328
usbfs_urb, 330
std::vector< AI16_DataPoint >
AIOUSB::AI16_DataPoint, 148
stdint.h
INT16_C, 844
INT16_MAX, 843
INT16_MIN, 843
INT32_C, 844
INT32_MAX, 843
INT32_MIN, 843
INT64_C, 844
INT64_MAX, 843
INT64_MIN, 843
INT8_C, 844
INT8_MAX, 843
INT8_MIN, 843
INT_FAST16_MAX, 844
INT_FAST16_MIN, 843
INT_FAST32_MAX, 844
INT_FAST32_MIN, 843
INT_FAST64_MAX, 844
INT_FAST64_MIN, 843
INT_FAST8_MAX, 844
INT_FAST8_MIN, 843
INT_LEAST16_MAX, 843
INT_LEAST16_MIN, 843
INT_LEAST32_MAX, 843
INT_LEAST32_MIN, 843
INT_LEAST64_MAX, 843
INT_LEAST64_MIN, 843
INT_LEAST8_MAX, 843
INT_LEAST8_MIN, 843
INTMAX_C, 844
INTMAX_MAX, 844
INTMAX_MIN, 844
INTPTR_MAX, 844
INTPTR_MIN, 844
int16_t, 845
int32_t, 845
int64_t, 845
int8_t, 845
int_fast16_t, 845
int_fast32_t, 845
int_fast64_t, 845
int_fast8_t, 845
int_least16_t, 845
int_least32_t, 845
int_least64_t, 845
int_least8_t, 845
intmax_t, 845
intptr_t, 844
PTRDIFF_MAX, 844
PTRDIFF_MIN, 844
ptrdiff_t, 845
SIG_ATOMIC_MAX, 844
SIG_ATOMIC_MIN, 844
SIZE_MAX, 844
UINT16_C, 844
UINT16_MAX, 843
UINT32_C, 844
UINT32_MAX, 843
UINT64_C, 844
UINT64_MAX, 843
UINT8_C, 844
UINT8_MAX, 843
UINT_FAST16_MAX, 844
UINT_FAST32_MAX, 844
UINT_FAST64_MAX, 844
UINT_FAST8_MAX, 844
UINT_LEAST16_MAX, 843
UINT_LEAST32_MAX, 843
UINT_LEAST64_MAX, 843
UINT_LEAST8_MAX, 843
UINTMAX_C, 844
UINTMAX_MAX, 844
UINTPTR_MAX, 844
uint16_t, 845
uint32_t, 845
uint64_t, 845
uint8_t, 845
uint_fast16_t, 845
uint_fast32_t, 845
uint_fast64_t, 845
uint_fast8_t, 845
uint_least16_t, 845
uint_least32_t, 845
uint_least64_t, 845
uint_least8_t, 845
uintmax_t, 845
uintptr_t, 845
WCHAR_MAX, 844
WCHAR_MIN, 844
WINT_MAX, 844
WINT_MIN, 844
wchar_t, 845
wctype_t, 845
wint_t, 845
Stop
RosterLooper, 267
USBRoster, 334
stop_scan
AIOCmd, 157
stop_scan_arg
AIOCmd, 157
stopClock
AIOUSB::CounterSubsystem, 206
AIOUSB::DIOStreamSubsystem, 227
str
htab_entry, 233
lookup, 260
strbuf
DIOBuf, 225
strbuf_size
DIOBuf, 225
stream_id
usbfs_urb, 330
usbi_transfer, 333
stream_test.c
find_ai_board, 866
main, 866
StreamingBlockSize
AIOUSBDevice, 167
stress.c
LIST_COUNT, 847
main, 847
string
cJSON, 198
hid_device_priv, 231
usb_raw_command, 310
string_index
hid_device_priv, 232
usb_raw_command, 310
StringArray, 268
AIOUSB::StringArray, 268
strrep
AIOChannelMask, 156

strrepsmall
AIOChannelMask, 156
strvalue
lookup, 260
sub_api
windows_device_priv, 338
submit_bulk_transfer
windows_usb_api_backend, 340
submit_control_transfer
windows_usb_api_backend, 340
submit_iso_transfer
windows_usb_api_backend, 340
submit_transfer
usbi_os_backend, 332
SubmitTransfer
USBDeviceHandle, 319
subsystem
AIOUSB::AI16_DataSet, 151
sum
aio_counts_converter, 153
sunos_backend
libusbi.h, 786
sunos_usb.c, 801
sunos_dev_handle_priv_t
sunos_usb.h, 802
sunos_dev_priv_t
sunos_usb.h, 802
sunos_device_handle_priv, 268
altsetting, 268
config_index, 268
dpriv, 268
eps, 268
sunos_device_priv, 268
cfgvalue, 269
dev_descr, 269
phypath, 269
raw_cfgdescr, 269
ugenpath, 269
sunos_ep_priv_t
sunos_usb.h, 802
sunos_transfer_priv, 269
aiocb, 269
transfer, 269
sunos_usb.c
sunos_backend, 801
sunos_usb.h
READ, 802
sunos_dev_handle_priv_t, 802
sunos_dev_priv_t, 802
sunos_ep_priv_t, 802
sunos_xfer_priv_t, 802
WRITE, 802
sunos_xfer_priv_t
sunos_usb.h, 802
SuperSpeed
windows_usbdk.h, 815
supportedProductIDs
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_CTR_15_Family, 289
AIOUSB::USB_DA12_8A_Family, 291
AIOUSB::USB_DA12_8E_Family, 294
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 300
AIOUSB::USB_DIO_Family, 302
Synchronous device I/O, 140
libusb_bulk_transfer, 140
libusb_control_transfer, 140
libusb_interrupt_transfer, 141
sys_path
darwin_cached_device, 214
sysfs_dir
linux_device_priv, 259

TERMINATED
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
TERMINATED_OVERRUN
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
TERMINATING
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
TERSE
AIOUSB_Properties.h, 655
ndkbuild/jni/AIOUSB_Properties.h, 656, 657
TEST_STATUS_ERROR
libusb_testlib.h, 846
TEST_STATUS_FAILURE
libusb_testlib.h, 846
TEST_STATUS_SKIP
libusb_testlib.h, 846
TEST_STATUS_SUCCESS
libusb_testlib.h, 846
THREAD_STATUS_begin
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
THREAD_STATUS_end
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
TIMEOUT_OPT
AIOCommandLine.h, 404
ndkbuild/jni/AIOCommandLine.h, 408
TAIL_Q_LIST
AIOList.h, 501, 502
ndkbuild/jni/AIOList.h, 503
TAIL_Q_LIST_TYPE
AIOList.h, 501, 502
ndkbuild/jni/AIOList.h, 503
TERSE_LOGGING
TestCaseSetup.h, 852
TEST_ADC_BulkPoll
TestCaseSetup, 275
THREAD_STATUS
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
THREADS_POSIX
android/config.h, 754
Xcode/config.h, 755
THROW_ERROR
TestCaseSetup.h, 852
THROW_IF_ERROR
TestCaseSetup, 274
TIMESPEC_IS_SET
libusbi.h, 783
TIMESPEC_TO_TIMEVAL
libusbi.h, 783
TRACE
TestCaseSetup.h, 852
TRACE_LEVEL
TestCaseSetup.h, 851
TRANSFER_CTX
libusbi.h, 784
TRIG_MODE_SCAN
AIOUSB::AnalogInputSubsystem, 185
TRIG_MODE_TIMER
AIOUSB::AnalogInputSubsystem, 185
TYPE
AIOFifo.h, 495
ndkbuild/jni/AIOFifo.h, 498
tag_USB_DK_DEVICE_ID, 269
targetSerialNumber
config_options, 199

tattr
AIOContinuousBuf, 160
test.c
capture_data, 860
fnd, 860
fp, 860
get_channel_range, 860
main, 860
process_cmd_line, 860
test_count
libusb_testlib_ctx, 254
test_fastscan.cpp
main, 866
test_names
libusb_testlib_ctx, 254
test_type
xusb.c, 767
TestCaseSetup, 272
~TestCaseSetup, 273
block_size, 275
CAL_CHANNEL, 275
calibration_enabled, 275
clock_speed, 275
configBlock, 275
counts, 275
dataBuf, 275
deviceFound, 275
DeviceIndex, 275
doBulkAcquire, 274
doBulkConfigBlock, 273
doCSVReadVoltages, 274
doCSVWithGetChannelV, 274
doCleanupAfterBulk, 274
doDACDirect, 274
doDACDirectSetup, 274
doDemonstrateReadVoltages, 274
doDisplayBulkResults, 275
doFastITScan, 274
doFastITScanSetup, 274
doGenericVendorWrite, 274
doGetBuffer, 274
doPreReadImmediateVoltages, 274
doPreSetup, 273
doScanSingleChannel, 274
doSetAutoCalibration, 273
doSomething, 273
doTestSetAutoCalibration, 274
doVerifyGroundCalibration, 273
doVerifyReferenceCalibration, 274
envGetDouble, 274
envGetInteger, 274
findDevice, 273
gainCodes, 275
getCounts, 275
getGainCodes, 275
getVolts, 275
MAX_CHANNELS, 275
maxcounts, 275
NUM_CHANNELS, 275
nameSize, 275
numCounters, 275
numDIOBytes, 275
number_oversamples, 275
productID, 275
resetCPU, 275
setCurrentDeviceIndex, 273
setMaxCount, 274
setupVoltageParameters, 275
TEST_ADC_BulkPoll, 275
THROW_IF_ERROR, 274
TestCaseSetup, 273
TestCaseSetup, 273
ThrowError, 274
volts, 275
writeBuffer, 274
TestCaseSetup.h
ALERT_LEVEL, 851
CHECK_RESULT, 852
DEBUG, 852
DEBUG_LEVEL, 851
DEF_CAL_CHANNEL, 852
DEF_MAX_CHANNELS, 852
DEF_NUM_CHANNELS, 852
ERROR, 852
ERROR_LEVEL, 851
FATAL, 852
FATAL_LEVEL, 851
INFO, 852
INFO_LEVEL, 851
LOG, 851
MAX_NAME_SIZE, 852
TERSE_LOGGING, 852
THROW_ERROR, 852
TRACE, 852
TRACE_LEVEL, 851
VERBOSE_LOGGING, 852
WARN_LEVEL, 851
testing
ADCConfigBlock, 146
AIOContinuousBuf, 161
AIOUSBDevice, 168
testlib.c
IGNORE_RETVAL, 848
INVALID_FD, 848
libusb_testlib_logf, 848
libusb_testlib_run_tests, 848
NULL_PATH, 848
testlibusb.c
main, 765
verbose, 765
thread_id
poll_windows.c, 798
threaded
AIOArgument, 155
threads_posix.c
usbi_cond_timedwait, 802
usbi_get_tid, 802
threads_posix.h
usbi_cond_broadcast, 803
usbi_cond_destroy, 803
usbi_cond_init, 803
usbi_cond_t, 803
usbi_cond_timedwait, 803
usbi_cond_wait, 803
usbi_get_tid, 803
usbi_mutex_destroy, 803
usbi_mutex_init, 803
usbi_mutex_lock, 803
usbi_mutex_static_lock, 803
usbi_mutex_static_t, 803
usbi_mutex_static_unlock, 803
usbi_mutex_t, 803
usbi_mutex_trylock, 803
usbi_mutex_unlock, 803
usbi_tls_key_create, 803
usbi_tls_key_delete, 803
usbi_tls_key_get, 803
usbi_tls_key_set, 803
usbi_tls_key_t, 803
threads_windows.c
usbi_cond_broadcast, 804
usbi_cond_destroy, 804
usbi_cond_init, 804
usbi_cond_timedwait, 804

usbi_cond_wait, 804
usbi_get_tid, 804
usbi_mutex_destroy, 804
usbi_mutex_init, 804
usbi_mutex_lock, 804
usbi_mutex_static_lock, 804
usbi_mutex_static_unlock, 804
usbi_mutex_trylock, 804
usbi_mutex_unlock, 804
usbi_tls_key_create, 804
usbi_tls_key_delete, 804
usbi_tls_key_get, 804
usbi_tls_key_set, 804
threads_windows.h
ETIMEDOUT, 805
usbi_cond_broadcast, 806
usbi_cond_destroy, 806
usbi_cond_init, 805
usbi_cond_t, 805
usbi_cond_timedwait, 806
usbi_cond_wait, 806
usbi_get_tid, 806
usbi_mutex_destroy, 805
usbi_mutex_init, 805
usbi_mutex_lock, 805
usbi_mutex_static_lock, 805
usbi_mutex_static_t, 805
usbi_mutex_static_unlock, 805
usbi_mutex_t, 805
usbi_mutex_trylock, 805
usbi_mutex_unlock, 805
usbi_tls_key_create, 806
usbi_tls_key_delete, 806
usbi_tls_key_get, 806
usbi_tls_key_set, 806
usbi_tls_key_t, 805
ThrowError
TestCaseSetup, 274
tid
usbi_cond_perthread, 331
timeStamp
AIOUSB::AI16_DataSet, 151
timeout
ADCConfigBlock, 146
AIOContinuousBuf, 161
AIOWDGConfig, 168
configuration, 200
libusb_transfer, 256
USBDevice, 313
usbfs_bulktransfer, 326
usbfs_ctrltransfer, 327
usbi_transfer, 333
timeout_flags
usbi_transfer, 333
timer_request, 276
event, 276
tp, 276
timersub
poll_windows.h, 799
timespec, 276
tv_nsec, 276
tv_sec, 276
tmp.c
BITS_PER_BYTE, 874
DEVICES_REQUIRED, 874
find_dio_96, 875
MAKE_MASK, 875
MASK_BYTES, 874
MAX_DIO_BYTES, 874
MAX_NAME_SIZE, 874
main, 875
PORT_A, 875
PORT_B, 874
PORT_C, 874
show_byte, 875
toString
AIOUSB::AI16_DataPoint, 148
total
ram_poke_context, 267
tp
timer_request, 276
transfer
aio_callback_args, 152
sunos_transfer_priv, 269
usb_raw_command, 311
TransferFailure
windows_usbdk.h, 815
TransferSuccess
windows_usbdk.h, 815
TransferSuccessAsync
windows_usbdk.h, 815
transfer_priv_size
usbi_os_backend, 333
TransferResult
tag_USB_DK_ISO_TRANSFER_RESULT, 271
windows_usbdk.h, 815
TransferType
tag_USB_DK_TRANSFER_REQUEST, 271
transferred
usbi_transfer, 333
TransfersThread
USBDeviceHandle, 319
TransfersWorker
USBDeviceHandle, 319
triggerMode
AIOUSB::AI16_DataSet, 151
AIOUSB::AnalogInputSubsystem, 187
Tristates
AIOUSBDevice, 166
DeviceProperties, 217
true
libusb_testlib.h, 846
windows_common.h, 809
xusb.c, 766
TrueFalse
AIOContinuousBuffer.c, 425
ndkbuild/jni/AIOContinuousBuffer.c, 434
tv_nsec
timespec, 276
tv_sec
timespec, 276
type
aio_ret_value, 154
AIOBuf, 155
AIOContinuousBuf, 161
cJSON, 198
configuration, 200
fx_known_device, 231
libusb_transfer, 255
usb_raw_command, 310
usbfs_urb, 330

u
USB_HUB_NAME_FIXED, 305
USB_NODE_INFORMATION, 307
USB descriptors
LIBUSB_CLASS_APPLICATION, 112
LIBUSB_CLASS_AUDIO, 111
LIBUSB_CLASS_COMM, 111
LIBUSB_CLASS_CONTENT_SECURITY, 112
LIBUSB_CLASS_DATA, 112
LIBUSB_CLASS_DIAGNOSTIC_DEVICE, 112
LIBUSB_CLASS_HID, 111
LIBUSB_CLASS_HUB, 112

LIBUSB_CLASS_IMAGE, 112
LIBUSB_CLASS_MASS_STORAGE, 112
LIBUSB_CLASS_PER_INTERFACE, 111
LIBUSB_CLASS_PERSONAL_HEALTHCARE, 112
LIBUSB_CLASS_PHYSICAL, 111
LIBUSB_CLASS_PRINTER, 112
LIBUSB_CLASS_PTP, 112
LIBUSB_CLASS_SMART_CARD, 112
LIBUSB_CLASS_VENDOR_SPEC, 112
LIBUSB_CLASS_VIDEO, 112
LIBUSB_CLASS_WIRELESS, 112
LIBUSB_DT_BOS, 112
LIBUSB_DT_CONFIG, 112
LIBUSB_DT_DEVICE, 112
LIBUSB_DT_DEVICE_CAPABILITY, 112
LIBUSB_DT_ENDPOINT, 112
LIBUSB_DT_HID, 112
LIBUSB_DT_HUB, 112
LIBUSB_DT_INTERFACE, 112
LIBUSB_DT_PHYSICAL, 112
LIBUSB_DT_REPORT, 112
LIBUSB_DT_SS_ENDPOINT_COMPANION, 112
LIBUSB_DT_STRING, 112
LIBUSB_DT_SUPERSPEED_HUB, 112
LIBUSB_ENDPOINT_IN, 112
LIBUSB_ENDPOINT_OUT, 112
LIBUSB_ISO_SYNC_TYPE_ADAPTIVE, 113
LIBUSB_ISO_SYNC_TYPE_ASYNC, 113
LIBUSB_ISO_SYNC_TYPE_NONE, 113
LIBUSB_ISO_SYNC_TYPE_SYNC, 113
LIBUSB_ISO_USAGE_TYPE_DATA, 113
LIBUSB_ISO_USAGE_TYPE_FEEDBACK, 113
LIBUSB_ISO_USAGE_TYPE_IMPLICIT, 113
LIBUSB_TRANSFER_TYPE_BULK, 112
LIBUSB_TRANSFER_TYPE_BULK_STREAM, 112
LIBUSB_TRANSFER_TYPE_CONTROL, 112
LIBUSB_TRANSFER_TYPE_INTERRUPT, 112
LIBUSB_TRANSFER_TYPE_ISOCHRONOUS, 112
USB-DIO-96/read_and_write_sample.c
NO_DEVICE_FOUND, 869
SUCCESS, 869
USB_ERROR, 869
USB_AI12_128
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_128A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_128E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_16
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_16A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_16E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_32
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_32A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_32E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64M
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64MA
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_64ME
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_96
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_96A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI12_96E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_128A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_128E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_16A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_16E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_32A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_32E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_64A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_64E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_64MA
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_64ME
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_96A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AI16_96E
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AIO12_128
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_128A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_128E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_16
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547

USB_AIO12_16A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_16E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_32
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_32A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_32E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64M
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64MA
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_64ME
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_96
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_96A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO12_96E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_128A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_128E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_16A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_16E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_32A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_32E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_64A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_64E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_64MA
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_64ME
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_96A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AIO16_96E
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_12
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_12A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_16
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_16A
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO12_4
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_4A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_8
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO12_8A
AIOTypes.h, 523, 525
ndkbuild/jni/AIOTypes.h, 545, 547
USB_AO16_12
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO16_12A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AO16_16
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AO16_16A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_AO16_4
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO16_4A
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO16_8
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO16_8A
AIOTypes.h, 522, 525
ndkbuild/jni/AIOTypes.h, 544, 547
USB_AO_ARB1
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_BULK_READ_ENDPOINT
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
USB_BULK_WRITE_ENDPOINT
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
USB_CTR_15
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_DA12_8A
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DA12_8A_REV_A
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545

USB_DA12_8E
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DI16A
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DI16A_REV_A1
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DI16A_REV_A2
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO16RO8
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_DIO24_CTR6
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO24DO12
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_DIO48DO24
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_DIO_16A
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO_16H
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO_32
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO_32I
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO_48
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DIO_96
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DO16A
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DO16A_REV_A1
AIOTypes.h, 521, 523
ndkbuild/jni/AIOTypes.h, 543, 545
USB_DO24
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_ERROR
USB-DIO-96/read_and_write_sample.c, 869
write_sample.c, 877
USB_IDIO_16
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IDIO_4
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IDIO_8
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IDO_16
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_II_16
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_II_16_OLD
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_II_8
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_II_8_OLD
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IIRO4_2SM
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IIRO4_COM
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USB_IIRO_16
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_IIRO_4
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_IIRO_8
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_READ_FROM_DEVICE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
USB_RO_16
AIOTypes.h, 521, 524
ndkbuild/jni/AIOTypes.h, 543, 546
USB_WRITE_TO_DEVICE
AIOTypes.h, 533
ndkbuild/jni/AIOTypes.h, 555
USBFS_URB_TYPE_BULK
linux_usbfs.h, 794
USBFS_URB_TYPE_CONTROL
linux_usbfs.h, 794
USBFS_URB_TYPE_INTERRUPT
linux_usbfs.h, 794
USBFS_URB_TYPE_ISO
linux_usbfs.h, 794
USBI_CLOCK_MONOTONIC
libusbi.h, 784
USBI_CLOCK_REALTIME
libusbi.h, 784
USBI_EVENT_POLLFDS_MODIFIED
libusbi.h, 784
USBI_EVENT_USER_INTERRUPT
libusbi.h, 784
USBI_TRANSFER_CANCELLING
libusbi.h, 784
USBI_TRANSFER_DEVICE_DISAPPEARED
libusbi.h, 784
USBI_TRANSFER_IN_FLIGHT
libusbi.h, 784
USBI_TRANSFER_OS_HANDLES_TIMEOUT
libusbi.h, 785
USBI_TRANSFER_TIMED_OUT
libusbi.h, 785
USBI_TRANSFER_TIMEOUT_HANDLED
libusbi.h, 785
USBP_II8IDO4A
AIOTypes.h, 522, 524
ndkbuild/jni/AIOTypes.h, 544, 546
USE_GENERIC
xusb.c, 767
USE_HID
xusb.c, 767
USE_PS3
xusb.c, 767
USE_SCSI
xusb.c, 767
USE_XBOX
xusb.c, 767
UCharArray, 276

AIOUSB::UCharArray, 276
UINT16_C
stdint.h, 844
UINT16_MAX
stdint.h, 843
UINT32_C
stdint.h, 844
UINT32_MAX
stdint.h, 843
UINT64_C
stdint.h, 844
UINT64_MAX
stdint.h, 843
UINT8_C
stdint.h, 844
UINT8_MAX
stdint.h, 843
UINT_FAST16_MAX
stdint.h, 844
UINT_FAST32_MAX
stdint.h, 844
UINT_FAST64_MAX
stdint.h, 844
UINT_FAST8_MAX
stdint.h, 844
UINT_LEAST16_MAX
stdint.h, 843
UINT_LEAST32_MAX
stdint.h, 843
UINT_LEAST64_MAX
stdint.h, 843
UINT_LEAST8_MAX
stdint.h, 843
UINTMAX_C
stdint.h, 844
UINTMAX_MAX
stdint.h, 844
UINTPTR_MAX
stdint.h, 844
UKW_CONTROL_HEADER, 276
bRequest, 277
bmRequestType, 277
wIndex, 277
wLength, 277
wValue, 277
UKW_DEVICE
wince_usb.h, 808
UKW_TF_NO_WAIT
wince_usb.h, 807
UNUSED
libusbi.h, 783
USAGE
windows_winusb.h, 829
USB descriptors, 110
libusb_class_code, 111
libusb_descriptor_type, 112
libusb_endpoint_direction, 112
libusb_free_bos_descriptor, 115
libusb_free_config_descriptor, 114
libusb_free_container_id_descriptor, 117
libusb_free_ss_endpoint_companion_descriptor, 114
libusb_free_ss_usb_device_capability_descriptor, 117
libusb_free_usb_2_0_extension_descriptor, 115
libusb_get_active_config_descriptor, 113
libusb_get_bos_descriptor, 115
libusb_get_config_descriptor, 113
libusb_get_config_descriptor_by_value, 114
libusb_get_container_id_descriptor, 117
libusb_get_device_descriptor, 113
libusb_get_ss_endpoint_companion_descriptor, 114
libusb_get_ss_usb_device_capability_descriptor, 115
libusb_get_string_descriptor_ascii, 117
libusb_get_usb_2_0_extension_descriptor, 115
libusb_iso_sync_type, 112
libusb_iso_usage_type, 113
libusb_transfer_type, 112
USB-AI16-16/sample.cpp
main, 863
USB-AO16-16/sample.cpp
main, 863
USB-DA12-8A/sample.cpp
main, 863
USB-DIO-16/sample.cpp
main, 864
USB-DIO-32/read_and_write_sample.c
main, 867
USB-DIO-96/read_and_write_sample.c
find_dio_96, 869
main, 869
USB_AI16_Family, 278
AIOUSB::AnalogInputSubsystem, 185
AIOUSB::CounterSubsystem, 207
AIOUSB::DigitalIOSubsystem, 223
AIOUSB::USB_AI16_Family, 279
USB_AIO16_Family, 280
AIOUSB::USB_AIO16_Family, 282
USB_AO16_Family, 283
AIOUSB::AO16_AnalogOutputSubsystem, 195
AIOUSB::DigitalIOSubsystem, 223
AIOUSB::USB_AO16_Family, 284
USB_API_COMPOSITE
windows_winusb.h, 824
USB_API_HID
windows_winusb.h, 824
USB_API_HUB
windows_winusb.h, 824
USB_API_MAX
windows_winusb.h, 824
USB_API_SET
windows_winusb.h, 824
USB_API_WINUSBX
windows_winusb.h, 824
USB_CTR_15_Family, 287
AIOUSB::CounterSubsystem, 207
USB_DA12_8A_Family, 289
AIOUSB::DA12_AnalogOutputSubsystem, 211
USB_DA12_8E_Family, 291
AIOUSB::DA12_AnalogOutputSubsystem, 211
USB_DIO_16_Family, 295
AIOUSB::DigitalIOSubsystem, 224
AIOUSB::DIOStreamSubsystem, 228
USB_DIO_32_Family, 298
AIOUSB::CounterSubsystem, 207
AIOUSB::DigitalIOSubsystem, 224
USB_DIO_Family, 300
AIOUSB::DigitalIOSubsystem, 223
AIOUSB::USB_DIO_Family, 301
USB_DK_DEVICE_ID
windows_usbdk.h, 814
USB_HUB_CAP_FLAGS, 303
HubIsRoot, 303
windows_winusb.h, 828
USB_HUB_CAPABILITIES, 303
USB_HUB_DESCRIPTOR, 304
bDescriptorLength, 304
bDescriptorType, 304
wHubCharacteristics, 304
windows_winusb.h, 827
USB_HUB_INFORMATION, 304
HubDescriptor, 304
USB_HUB_NAME_FIXED, 305

node, 305
root, 305
u, 305
USB_HUB_NODE
windows_winusb.h, 827, 830
USB_MAXCONFIG
libusbi.h, 782
USB_MAXENDPOINTS
libusbi.h, 782
USB_MAXINTERFACES
libusbi.h, 782
USB_NODE_INFORMATION, 307
u, 307
USB_PIPE_INFO, 307
EndpointDescriptor, 308
ScheduleOffset, 308
windows_winusb.h, 827
USB_PROTOCOLS
windows_winusb.h, 827
USB_ROOT_HUB_NAME, 311
ActualLength, 311
RootHubName, 311
windows_winusb.h, 827
USB_RQ
dpfp.c, 756
dpfp_threaded.c, 757
USBD_PIPE_TYPE
windows_winusb.h, 828, 830
USBDEV
openbsd_usb.c, 796
USBDK_ABORT_PIPE
windows_usbdk.h, 815
USBDK_READ_PIPE
windows_usbdk.h, 815
USBDK_RESET_DEVICE
windows_usbdk.h, 815
USBDK_RESET_PIPE
windows_usbdk.h, 815
USBDK_WRITE_PIPE
windows_usbdk.h, 815
USBDevice, 311
~USBDevice, 313
ActiveConfiguration, 313
ActiveConfigurationIndex, 313
altset, 314
CheckInterfacesFree, 313
ClaimInterface, 313
conf, 313
ConfigurationDescriptor, 313
CountConfigurations, 313
debug, 313
Descriptor, 313
device, 313
deviceDesc, 313
deviceHandle, 313
EndpointToIndex, 313
EndpointToInterface, 313
fActiveConfiguration, 314
fClaimedInterfaces, 314
fConfigToIndex, 314
fConfigurationDescriptors, 314
fDeviceDescriptor, 314
fEndpointToIndex, 314
fEndpointToInterface, 314
fInitCheck, 314
fPath, 314
iface, 313
InitCheck, 313
Initialise, 313
Location, 313
ndkbuild/jni/USBDevice.h, 751
origconf, 314
ReleaseInterface, 313
SetActiveConfiguration, 313
timeout, 313
USBDevice, 313
usb_bulk_transfer, 313
usb_control_transfer, 313
usb_get_config, 313
usb_put_config, 313
usb_request, 313
usb_reset_device, 313
USBDevice, 313
USBDevice.h, 753
usblp_attached, 313
verbose, 313
USBDevice.c
AddAllACCESUSBDevices, 750
AddDevice, 750
CopyUSBDevice, 750
DeleteUSBDevice, 750
DeleteUSBDevices, 750
get_usb_device, 750
InitializeUSBDevice, 750
NewUSBDevice, 750
USBDeviceClose, 750
USBDeviceFetchADCConfigBlock, 750
USBDeviceGetIdProduct, 750
USBDeviceGetUSBDeviceHandle, 750
USBDevicePutADCConfigBlock, 750
USBDeviceSetDebug, 750
usb_bulk_transfer, 750
usb_control_transfer, 750
usb_request, 750
usb_reset_device, 750
USBDevice.h
AddAllACCESUSBDevices, 753
CopyUSBDevice, 753
DeleteUSBDevice, 753
DeleteUSBDevices, 753
get_usb_device, 754
INTERNAL_METHOD, 753
InitializeUSBDevice, 753
LIBUSBArgs, 753
NewUSBDevice, 753
USBDevice, 753
USBDeviceClose, 753
USBDeviceFetchADCConfigBlock, 753
USBDeviceGetIdProduct, 753
USBDeviceGetUSBDeviceHandle, 754
USBDevicePutADCConfigBlock, 753
usb_bulk_transfer, 753
usb_control_transfer, 753
usb_request, 753
usb_reset_device, 754
USBDeviceArray, 314
AIOUSB::USBDeviceArray, 314
USBDeviceBase, 314
AIOUSB::USBDeviceBase, 315
USBDeviceClose
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 748
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 753
USBDeviceFetchADCConfigBlock
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 753
USBDeviceGetIdProduct
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749

ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 753
USBDeviceGetUSBDeviceHandle
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 754
USBDeviceHandle, 318
~USBDeviceHandle, 319
CancelTransfer, 319
ClaimInterface, 319
fClaimedInterfaces, 319
fInitCheck, 319
fRawFD, 319
fTransfers, 319
fTransfersLock, 319
fTransfersSem, 319
fTransfersThread, 319
fUSBDevice, 319
InitCheck, 319
ReleaseInterface, 319
SetAltSetting, 319
SetConfiguration, 319
SubmitTransfer, 319
TransfersThread, 319
TransfersWorker, 319
USBDeviceHandle, 319
USBDeviceHandle, 319
USBDeviceManager, 320
AIOUSB::USB_AI16_Family, 280
AIOUSB::USB_AIO16_Family, 283
AIOUSB::USB_AO16_Family, 285
AIOUSB::USB_CTR_15_Family, 289
AIOUSB::USB_DA12_8A_Family, 291
AIOUSB::USB_DA12_8E_Family, 294
AIOUSB::USB_DIO_16_Family, 297
AIOUSB::USB_DIO_32_Family, 300
AIOUSB::USB_DIO_Family, 302
AIOUSB::USBDeviceBase, 318
AIOUSB::USBDeviceManager, 321
USBDevicePutADCConfigBlock
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice.c, 750
USBDevice.h, 753
USBDeviceSetDebug
ndkbuild/jni/USBDevice.c, 749
USBDevice.c, 750
USBFS_MAXDRIVERNAME
linux_usbfs.h, 793
USBI_GET_CONTEXT
libusbi.h, 783
USBI_LOG_LINE_END
libusbi.h, 783
USBI_MAX_LOG_LEN
libusbi.h, 782
USBRoster, 334
~USBRoster, 334
fLooper, 334
Start, 334
Stop, 334
USBRoster, 334
USBRoster, 334
USBTransfer, 334
~USBTransfer, 335
Do, 335
fCancelled, 335
fLibusbTransfer, 335
fStatusLock, 335
fUSBDevice, 335
fUsbiTransfer, 335
IsCancelled, 335
SetCancelled, 335
USBTransfer, 335
UsbiTransfer, 335
USBTransfer, 335
UShortArray, 335
AIOUSB::UShortArray, 335
ugenpath
sunos_device_priv, 269
uint16_t
stdint.h, 845
uint32_t
stdint.h, 845
uint64_t
stdint.h, 845
uint8_t
stdint.h, 845
uint_fast16_t
stdint.h, 845
uint_fast32_t
stdint.h, 845
uint_fast64_t
stdint.h, 845
uint_fast8_t
stdint.h, 845
uint_least16_t
stdint.h, 845
uint_least32_t
stdint.h, 845
uint_least64_t
stdint.h, 845
uint_least8_t
stdint.h, 845
uintmax_t
stdint.h, 845
uintptr_t
stdint.h, 845
unit_size
aio_counts_converter, 153
AIOContinuousBuf, 160
urbs
linux_transfer_priv, 259
Usage
HIDP_CAPS, 233
UsagePage
HIDP_CAPS, 233
UsbHub
windows_winusb.h, 830
UsbMIParent
windows_winusb.h, 830
usb_api_backend
windows_winusb.c, 818
windows_winusb.h, 831
usb_bulk_transfer
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice, 313
USBDevice.c, 750
USBDevice.h, 753
usb_bulk_xfer
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_close
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_control_transfer
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752

USBDevice, 313
USBDevice.c, 750
USBDevice.h, 753
usb_control_xfer
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_descriptor_header, 294
bDescriptorType, 294
bLength, 294
usb_device
AIOUSBDevice, 166
usb_devs
libusb_context, 239
usb_devs_lock
libusb_context, 239
usb_free_devices
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_get_config
USBDevice, 313
usb_get_devices
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_interface
windows_device_priv, 338
xusb.c, 766
usb_open
AIOUSB_USB.h, 658
ndkbuild/jni/AIOUSB_USB.h, 659
usb_put_config
USBDevice, 313
usb_raw_command, 308
alternate, 310
alternate_index, 310
alternate_info, 310
config, 310
config_index, 310
control, 311
data, 310
descriptor, 310
device, 310
endpoint, 310, 311
endpoint_etc, 310
endpoint_index, 310
generic, 310
generic_etc, 310
generic_index, 310
index, 310
interface, 310, 311
interface_etc, 310
interface_index, 310
isochronous, 311
language_id, 310
length, 310, 311
packet_count, 311
packet_descriptors, 311
request, 310
request_type, 310
status, 310
string, 310
string_index, 310
transfer, 311
type, 310
value, 310
version, 310
usb_raw_command_id
haiku_usb_raw.h, 789
usb_raw_command_status
haiku_usb_raw.h, 789
usb_request
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice, 313
USBDevice.c, 750
USBDevice.h, 753
usb_reset_device
libaiousb.h, 746
ndkbuild/jni/USBDevice.c, 749
ndkbuild/jni/USBDevice.h, 752
USBDevice, 313
USBDevice.c, 750
USBDevice.h, 754
UsbdPipeTypeBulk
windows_winusb.h, 830
UsbdPipeTypeControl
windows_winusb.h, 830
UsbdPipeTypeInterrupt
windows_winusb.h, 830
UsbdPipeTypeIsochronous
windows_winusb.h, 830
UsbdStatus
tag_USB_DK_GEN_TRANSFER_RESULT, 270
usbdk_backend
libusbi.h, 785
usbfs_bulktransfer, 326
data, 326
ep, 326
len, 326
timeout, 326
usbfs_connectinfo, 326
devnum, 326
slow, 326
usbfs_ctrltransfer, 326
bRequest, 327
bmRequestType, 327
data, 327
timeout, 327
wIndex, 327
wLength, 327
wValue, 327
usbfs_disconnect_claim, 327
driver, 327
flags, 327
interface, 327
usbfs_getdriver, 327
driver, 327
interface, 327
usbfs_hub_portinfo, 328
numports, 328
port, 328
usbfs_ioctl, 328
data, 328
ifno, 328
ioctl_code, 328
usbfs_iso_packet_desc, 328
actual_length, 328
length, 328
status, 328
usbfs_setinterface, 329
altsetting, 329
interface, 329
usbfs_streams, 329
eps, 329
num_eps, 329
num_streams, 329
usbfs_urb, 329
actual_length, 330
buffer, 330
buffer_length, 330
endpoint, 330
error_count, 330
flags, 330
iso_frame_desc, 330

number_of_packets, 330
signr, 330
start_frame, 330
status, 330
stream_id, 330
type, 330
usercontext, 330
usbfs_urb_type
linux_usbfs.h, 794
usbi_add_pollfd
io.c, 773
libusbi.h, 785
usbi_alloc_device
core.c, 769
libusbi.h, 785
usbi_backend
libusbi.h, 785
usbi_clear_event
core.c, 769
libusbi.h, 785
usbi_close
poll_posix.h, 796
poll_windows.c, 798
poll_windows.h, 800
usbi_cond, 330
not_waiting, 330
waiters, 330
usbi_cond_broadcast
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_cond_destroy
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_cond_init
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_cond_perthread, 330
event, 331
list, 331
tid, 331
usbi_cond_t
threads_posix.h, 803
threads_windows.h, 805
usbi_cond_timedwait
threads_posix.c, 802
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_cond_wait
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_connect_device
core.c, 769
libusbi.h, 785
usbi_create_fd
poll_windows.c, 797
poll_windows.h, 800
usbi_dbg
libusbi.h, 783
usbi_default_context
core.c, 769
libusbi.h, 785
usbi_device_cache_descriptor
descriptor.c, 771
libusbi.h, 785
usbi_disconnect_device
core.c, 769
libusbi.h, 785
usbi_end_event_handling
libusbi.h, 784
usbi_err
libusbi.h, 783
usbi_event_flags
libusbi.h, 784
usbi_free_fd
poll_windows.c, 797
poll_windows.h, 800
usbi_get_config_index_by_value
descriptor.c, 771
libusbi.h, 785
usbi_get_device_by_session_id
core.c, 769
libusbi.h, 785
usbi_get_tid
threads_posix.c, 802
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_handle_disconnect
io.c, 773
libusbi.h, 785
usbi_handle_transfer_cancellation
io.c, 773
libusbi.h, 785
usbi_handle_transfer_completion
io.c, 773
libusbi.h, 785
usbi_handling_events
libusbi.h, 784
usbi_hotplug_deregister_all
hotplug.c, 771
hotplug.h, 772
usbi_hotplug_match
hotplug.c, 771
hotplug.h, 772
usbi_hotplug_notification
hotplug.c, 771
hotplug.h, 772
usbi_info
libusbi.h, 783
usbi_io_exit
io.c, 773
libusbi.h, 785
usbi_io_init
io.c, 773
libusbi.h, 785
usbi_log
core.c, 769
libusbi.h, 785
usbi_log_v
core.c, 769
libusbi.h, 785
usbi_mutex_destroy
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_init
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_lock
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_static_lock
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_static_t
threads_posix.h, 803

threads_windows.h, 805
usbi_mutex_static_unlock
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_t
threads_posix.h, 803
threads_windows.h, 805
usbi_mutex_trylock
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_mutex_unlock
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 805
usbi_os_backend, 331
alloc_streams, 332
attach_kernel_driver, 332
cancel_transfer, 332
caps, 332
claim_interface, 332
clear_halt, 332
clear_transfer_priv, 332
clock_gettime, 332
close, 332
destroy_device, 332
detach_kernel_driver, 332
dev_mem_alloc, 332
dev_mem_free, 332
device_handle_priv_size, 333
device_priv_size, 333
exit, 332
free_streams, 332
get_active_config_descriptor, 332
get_config_descriptor, 332
get_config_descriptor_by_value, 332
get_configuration, 332
get_device_descriptor, 332
get_device_list, 332
handle_events, 332
handle_transfer_completion, 332
hotplug_poll, 332
init, 332
kernel_driver_active, 332
name, 332
open, 332
release_interface, 332
reset_device, 332
set_configuration, 332
set_interface_altsetting, 332
submit_transfer, 332
transfer_priv_size, 333
usbi_parse_descriptor
descriptor.c, 771
libusbi.h, 785
usbi_pending_events
libusbi.h, 784
usbi_pipe
poll_posix.c, 796
poll_posix.h, 797
poll_windows.c, 797
poll_windows.h, 800
usbi_poll
poll_posix.h, 796
poll_windows.c, 798
poll_windows.h, 800
usbi_pollfd, 333
list, 333
pollfd, 333
usbi_read
poll_posix.h, 796
poll_windows.c, 798
poll_windows.h, 800
usbi_remove_pollfd
io.c, 773
libusbi.h, 785
usbi_sanitize_device
core.c, 769
libusbi.h, 785
usbi_signal_event
core.c, 769
libusbi.h, 785
usbi_signal_transfer_completion
io.c, 773
libusbi.h, 785
usbi_start_event_handling
libusbi.h, 784
usbi_tls_key_create
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_tls_key_delete
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_tls_key_get
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_tls_key_set
threads_posix.h, 803
threads_windows.c, 804
threads_windows.h, 806
usbi_tls_key_t
threads_posix.h, 803
threads_windows.h, 805
usbi_transfer, 333
completed_list, 333
list, 333
lock, 334
num_iso_packets, 333
state_flags, 333
stream_id, 333
timeout, 333
timeout_flags, 333
transferred, 333
usbi_transfer_state_flags
libusbi.h, 784
usbi_transfer_timeout_flags
libusbi.h, 784
usbi_using_timerfd
libusbi.h, 784
usbi_warn
libusbi.h, 783
usbi_write
poll_posix.h, 796
poll_windows.c, 798
poll_windows.h, 800
UsbiTransfer
USBTransfer, 335
usblp_attached
USBDevice, 313
Use_Duplicate_Handles
poll_windows.c, 797
use_maxcount
options, 263
used
htab_entry, 233
user_data
libusb_hotplug_callback, 247
libusb_transfer, 256
usercontext
usbfs_urb, 330

uses_report_ids
hid_device_priv, 231
Ushort_Array
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
ushort_array, 335
size, 335

VENDOR_REQUEST_begin
AIOTypes.h, 531
ndkbuild/jni/AIOTypes.h, 553
VENDOR_REQUEST_end
AIOTypes.h, 531, 532
ndkbuild/jni/AIOTypes.h, 553, 554
VALID_ENUM
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
VALID_PRODUCT
AIOTypes.h, 518
ndkbuild/jni/AIOTypes.h, 540
VENDOR_REQUEST
AIOTypes.h, 530
ndkbuild/jni/AIOTypes.h, 552
VERBOSE_LOGGING
TestCaseSetup.h, 852
VERSION_DATE
AIOUSB::USBDeviceManager, 325
VERSION_NUMBER
AIOUSB::USBDeviceManager, 325
val
option, 262
valid
AIOUSBDevice, 168
valid_config_block
AIOUSB_ADC.c, 562
ndkbuild/jni/AIOUSB_ADC.c, 575
value
lookup, 260
usb_raw_command, 310
WINUSB_SETUP_PACKET, 342
valuedouble
cJSON, 198
valueint
cJSON, 198
valuestring
cJSON, 198
vendor_id
libusb_hotplug_callback, 247
VendorID
HIDD_ATTRIBUTES, 232
verbose
AIOCommandLineOptions, 158
ezusb.c, 759
ezusb.h, 760
libusb_testlib_ctx, 254
opts, 264
testlibusb.c, 765
USBDevice, 313
version
usb_raw_command, 310
version.h
LIBUSB_MICRO, 832
LIBUSB_MINOR, 832
LIBUSB_RC, 832
version_nano.h
LIBUSB_NANO, 832
VersionNumber
HIDD_ATTRIBUTES, 232
vid
fx_known_device, 231
hid_device_priv, 231
void
Asynchronous device I/O, 130
volts
AIOUSB::OutputVoltagePoint, 265
TestCaseSetup, 275
voltsToCounts
AIOUSB::AnalogInputSubsystem, 184, 185
AIOUSB::AnalogIORange, 190
AIOUSB::AO16_AnalogOutputSubsystem, 194
AIOUSB::DA12_AnalogOutputSubsystem, 211

WDGVals_begin
AIOUSB_WDG.h, 663
ndkbuild/jni/AIOUSB_WDG.h, 664
WDGVals_end
AIOUSB_WDG.h, 663
ndkbuild/jni/AIOUSB_WDG.h, 664
WINDOWS_2003
poll_windows.h, 800
WINDOWS_7
poll_windows.h, 800
WINDOWS_8
poll_windows.h, 800
WINDOWS_8_1_OR_LATER
poll_windows.h, 800
WINDOWS_CE
poll_windows.h, 800
WINDOWS_MAX
poll_windows.h, 800
WINDOWS_UNDEFINED
poll_windows.h, 800
WINDOWS_UNSUPPORTED
poll_windows.h, 800
WINDOWS_VISTA
poll_windows.h, 800
WINDOWS_XP
poll_windows.h, 800
WITH_DATA
AIOTypes.h, 520
ndkbuild/jni/AIOTypes.h, 542
WARN_LEVEL
TestCaseSetup.h, 851
wBytesPerInterval
libusb_ss_endpoint_companion_descriptor, 252
WCHAR_MAX
stdint.h, 844
WCHAR_MIN
stdint.h, 844
wClassDescriptorLength
libusb_hid_descriptor, 246
WDG_GetStatus
AIOUSB_WDG.c, 659
AIOUSB_WDG.h, 663
libaiousb.h, 745
ndkbuild/jni/AIOUSB_WDG.c, 662
ndkbuild/jni/AIOUSB_WDG.h, 665
WDG_Pet
AIOUSB_WDG.c, 661
AIOUSB_WDG.h, 663
libaiousb.h, 745
ndkbuild/jni/AIOUSB_WDG.c, 662
ndkbuild/jni/AIOUSB_WDG.h, 665
WDG_SetConfig
AIOUSB_WDG.c, 659
AIOUSB_WDG.h, 663
libaiousb.h, 745
ndkbuild/jni/AIOUSB_WDG.c, 661
ndkbuild/jni/AIOUSB_WDG.h, 664
WDGBytes
AIOUSBDevice, 167
WDGVals
AIOUSB_WDG.h, 663
ndkbuild/jni/AIOUSB_WDG.h, 664

wHubCharacteristics
USB_HUB_DESCRIPTOR, 304
WINT_MAX
stdint.h, 844
WINT_MIN
stdint.h, 844
WINUSB_SETUP_PACKET, 342
index, 342
length, 342
request, 342
value, 342
WINUSBX_DRV_NAMES
windows_winusb.h, 824
wIndex
libusb_control_setup, 241
UKW_CONTROL_HEADER, 277
USB_CONFIGURATION_DESCRIPTOR_SHORT, 286
USB_DESCRIPTOR_REQUEST, 295
usbfs_ctrltransfer, 327
wLength
libusb_control_setup, 241
UKW_CONTROL_HEADER, 277
USB_CONFIGURATION_DESCRIPTOR_SHORT, 287
USB_DESCRIPTOR_REQUEST, 295
usbfs_ctrltransfer, 327
WM_TIMER_EXIT
windows_nt_common.c, 812
WM_TIMER_REQUEST
windows_nt_common.c, 812
wMaxPacketSize
libusb_endpoint_descriptor, 245
USB_ENDPOINT_DESCRIPTOR, 302
WRITE
sunos_usb.h, 802
wSpeedSupported
libusb_ss_usb_device_capability_descriptor, 253
wTotalLength
libusb_bos_descriptor, 235
libusb_config_descriptor, 237
USB_CONFIGURATION_DESCRIPTOR, 286
wValue
libusb_control_setup, 241
UKW_CONTROL_HEADER, 277
USB_CONFIGURATION_DESCRIPTOR_SHORT, 286
USB_DESCRIPTOR_REQUEST, 295
usbfs_ctrltransfer, 327
waiters
usbi_cond, 330
WatchedEntry, 336
~WatchedEntry, 336
EntryCreated, 336
EntryRemoved, 336
fDevice, 336
fEntries, 336
fInitCheck, 336
fIsDirectory, 336
fLink, 336
fMessenger, 336
fNode, 336
InitCheck, 336
WatchedEntry, 336
WatchedEntry, 336
wchar_t
stdint.h, 845
wchar_to_utf8_ms
windows_common.h, 810
wctype_t
stdint.h, 845
wdgbuf
AIOWDGConfig, 168
what
Error, 230
WinUSBX_Set
windows_winusb.c, 817
WinUsb_AbortPipe_t
windows_winusb.h, 828
WinUsb_ControlTransfer_t
windows_winusb.h, 828
WinUsb_FlushPipe_t
windows_winusb.h, 828
WinUsb_Free_t
windows_winusb.h, 828
WinUsb_GetAssociatedInterface_t
windows_winusb.h, 828
WinUsb_GetCurrentAlternateSetting_t
windows_winusb.h, 828
WinUsb_GetDescriptor_t
windows_winusb.h, 828
WinUsb_GetOverlappedResult_t
windows_winusb.h, 828
WinUsb_GetPipePolicy_t
windows_winusb.h, 828
WinUsb_GetPowerPolicy_t
windows_winusb.h, 828
WinUsb_Initialize_t
windows_winusb.h, 828
WinUsb_QueryDeviceInformation_t
windows_winusb.h, 828
WinUsb_QueryInterfaceSettings_t
windows_winusb.h, 828
WinUsb_QueryPipe_t
windows_winusb.h, 828
WinUsb_ReadPipe_t
windows_winusb.h, 828
WinUsb_ResetDevice_t
windows_winusb.h, 829
WinUsb_ResetPipe_t
windows_winusb.h, 828
WinUsb_SetCurrentAlternateSetting_t
windows_winusb.h, 829
WinUsb_SetPipePolicy_t
windows_winusb.h, 829
WinUsb_SetPowerPolicy_t
windows_winusb.h, 829
WinUsb_WritePipe_t
windows_winusb.h, 829
wince_backend
libusbi.h, 786
wince_usb.c, 806
wince_device_priv, 336
desc, 337
dev, 337
wince_transfer_priv, 337
interface_number, 337
pollable_fd, 337
wince_usb.c
epoch_time, 806
wince_backend, 806
windows_version, 806
wince_usb.h
DLL_DECLARE_FUNC, 808, 809
LPUKW_DEVICE, 808
MAX_DEVICE_COUNT, 807
PUKW_DEVICE, 808
STATUS_HALT_FLAG, 808
UKW_DEVICE, 808
UKW_TF_NO_WAIT, 807
windows_usbdk.h
BulkTransferType, 816
ControlTransferType, 816
FullSpeed, 815

HighSpeed, 815
IntertuptTransferType, 816
IsochronousTransferType, 816
LowSpeed, 815
NoSpeed, 815
SuperSpeed, 815
TransferFailure, 815
TransferSuccess, 815
TransferSuccessAsync, 815
windows_winusb.h
DeviceCausedOvercurrent, 830
DeviceConnected, 829
DeviceFailedEnumeration, 829
DeviceGeneralFailure, 830
DeviceHubNestedTooDeeply, 830
DeviceInLegacyHub, 830
DeviceNotEnoughBandwidth, 830
DeviceNotEnoughPower, 830
HID_REPORT_TYPE_FEATURE, 829
HID_REPORT_TYPE_INPUT, 829
HID_REPORT_TYPE_OUTPUT, 829
HID_REQ_GET_IDLE, 829
HID_REQ_GET_PROTOCOL, 829
HID_REQ_GET_REPORT, 829
HID_REQ_SET_IDLE, 829
HID_REQ_SET_PROTOCOL, 829
HID_REQ_SET_REPORT, 829
NoDeviceConnected, 829
UsbHub, 830
UsbMIParent, 830
UsbdPipeTypeBulk, 830
UsbdPipeTypeControl, 830
UsbdPipeTypeInterrupt, 830
UsbdPipeTypeIsochronous, 830
windows_backend
libusbi.h, 785
windows_winusb.c, 818
windows_clear_transfer_priv
windows_nt_common.h, 813
windows_winusb.c, 817
windows_clock_gettime
windows_nt_common.c, 812
windows_nt_common.h, 813
windows_common.h
ARRAYSIZE, 810
bool, 809
DLL_DECLARE_FUNC, 810
DLL_FREE_HANDLE, 810
DLL_GET_HANDLE, 810
DLL_LOAD_FUNC, 811
DLL_LOAD_LIBRARY, 810
DLL_STRINGIFY, 810
ERR_BUFFER_SIZE, 810
false, 809
safe_closehandle, 809
safe_free, 809
safe_min, 809
safe_sprintf, 810
safe_stprintf, 810
safe_strcat, 810
safe_strcmp, 810
safe_strcp, 809
safe_strcpy, 809
safe_stricmp, 810
safe_strlen, 810
safe_strncat, 810
safe_strncmp, 810
safe_tcslen, 810
safe_unref_device, 810
true, 809
wchar_to_utf8_ms, 810
windows_common_exit
windows_nt_common.c, 812
windows_nt_common.h, 813
windows_common_init
windows_nt_common.c, 812
windows_nt_common.h, 813
windows_copy_transfer_data
windows_nt_common.h, 813
windows_winusb.c, 817
windows_destroy_clock
windows_nt_common.c, 812
windows_device_handle_priv, 337
active_interface, 337
autoclaim_count, 337
interface_handle, 337
windows_device_priv, 337
active_config, 338
apib, 338
config_descriptor, 338
depth, 338
dev_descriptor, 338
endpoint, 338
hid, 338
nb_endpoints, 338
parent_dev, 338
path, 338
port, 338
restricted_functionality, 338
sub_api, 338
usb_interface, 338
windows_get_fd
windows_nt_common.h, 813
windows_winusb.c, 817
windows_get_overlapped_result
windows_nt_common.h, 813
windows_winusb.c, 817
windows_handle_callback
windows_nt_common.c, 812
windows_nt_common.h, 813
windows_handle_events
windows_nt_common.c, 812
windows_nt_common.h, 813
windows_nt_common.c
epoch_time, 812
HTAB_SIZE, 812
htab_entry, 812
htab_hash, 812
WM_TIMER_EXIT, 812
windows_clock_gettime, 812
windows_common_exit, 812
windows_common_init, 812
windows_destroy_clock, 812
windows_handle_callback, 812
windows_handle_events, 812
windows_nt_common.h
htab_hash, 813
windows_clear_transfer_priv, 813
windows_clock_gettime, 813
windows_common_exit, 813
windows_common_init, 813
windows_copy_transfer_data, 813
windows_get_fd, 813
windows_get_overlapped_result, 813
windows_handle_callback, 813
windows_handle_events, 813
windows_transfer_priv, 338
hid_buffer, 339
hid_dest, 339
hid_expected_size, 339
interface_number, 339
pollable_fd, 339
windows_usb_api_backend, 339
abort_control, 340

abort_transfers, 340
claim_interface, 339
clear_halt, 340
close, 339
configure_endpoints, 339
copy_transfer_data, 340
designation, 339
driver_name_list, 339
exit, 339
id, 339
init, 339
nb_driver_names, 339
open, 339
release_interface, 339
reset_device, 340
set_interface_altsetting, 339
submit_bulk_transfer, 340
submit_control_transfer, 340
submit_iso_transfer, 340
windows_usbdk.h
TransferResult, 815
USB_DK_DEVICE_ID, 814
USBDK_ABORT_PIPE, 815
USBDK_READ_PIPE, 815
USBDK_RESET_PIPE, 815
USBDK_WRITE_PIPE, 815
windows_version
poll_windows.h, 800
wince_usb.c, 806
windows_winusb.c, 817
windows_winusb.c
DEV_PASS, 817
GEN_PASS, 817
HANDLE_VALID, 816
HCD_PASS, 817
HID_PASS, 817
HUB_PASS, 817
LOOP_BREAK, 817
LOOP_CHECK, 816
MAX_ENUM_GUIDS, 817
usb_api_backend, 818
WinUSBX_Set, 817
windows_backend, 818
windows_clear_transfer_priv, 817
windows_copy_transfer_data, 817
windows_get_fd, 817
windows_get_overlapped_result, 817
windows_version, 817
windows_winusb.h
AUTO_CLEAR_STALL, 826
AUTO_FLUSH, 826
AUTO_SUSPEND, 826
CONFIGRET, 827
CR_SUCCESS, 825
CTL_CODE, 825
DEVICE_SPEED, 826
DEVINST, 826
DEVNODE, 826
DLL_DECLARE_FUNC, 830, 831
FILE_ANY_ACCESS, 825
FILE_DEVICE_USB, 825
FullSpeed, 826
HID_CTL_CODE, 824
HID_IN_CTL_CODE, 824
HID_IN_EP, 824
HID_OUT_CTL_CODE, 824
HID_OUT_EP, 824
HIDP_REPORT_TYPE, 829
HIDP_VALUE_CAPS, 829
HighSpeed, 826
INTERFACE_CLAIMED, 823
KLIB_VERSION, 829
KUSB_FNID, 829
LIBUSB_COMPLETED, 824
LIBUSB_REQ_IN, 824
LIBUSB_REQ_OUT, 824
LIBUSB_REQ_TYPE, 824
LIST_SEPARATOR, 823
LibK_GetProcAddress_t, 829
LibK_GetVersion_t, 829
libusb_hid_report_type, 829
libusb_hid_request_type, 829
LowSpeed, 826
MAX_KEY_LENGTH, 823
MAX_PATH_LENGTH, 823
MAX_USB_DEVICES, 823
METHOD_BUFFERED, 825
PDEVINST, 826
PDEVNODE, 826
PHIDD_ATTRIBUTES, 829
PHIDP_CAPS, 829
PHIDP_VALUE_CAPS, 829
PKLIB_VERSION, 829
PUSB_PIPE_INFO, 827
PUSB_PROTOCOLS, 827
RAW_IO, 826
RETURN_TYPE, 826
SPDRP_ADDRESS, 823
SUB_API_LIBUSB0, 824
SUB_API_LIBUSBK, 824
SUB_API_MAX, 824
SUB_API_NOTSET, 824
SUB_API_WINUSB, 824
SUSPEND_DELAY, 826
USAGE, 829
USB_API_COMPOSITE, 824
USB_API_HID, 824
USB_API_HUB, 824
USB_API_MAX, 824
USB_API_SET, 824
USB_API_WINUSBX, 824
USB_HUB_NODE, 827, 830
USB_PIPE_INFO, 827
USB_PROTOCOLS, 827
USBD_PIPE_TYPE, 828, 830
usb_api_backend, 831
WINUSBX_DRV_NAMES, 824
WinUsb_AbortPipe_t, 828
WinUsb_ControlTransfer_t, 828
WinUsb_FlushPipe_t, 828
WinUsb_Free_t, 828
WinUsb_GetAssociatedInterface_t, 828
WinUsb_GetCurrentAlternateSetting_t, 828
WinUsb_GetDescriptor_t, 828
WinUsb_GetOverlappedResult_t, 828
WinUsb_GetPipePolicy_t, 828
WinUsb_GetPowerPolicy_t, 828
WinUsb_Initialize_t, 828
WinUsb_QueryDeviceInformation_t, 828
WinUsb_QueryInterfaceSettings_t, 828
WinUsb_QueryPipe_t, 828
WinUsb_ReadPipe_t, 828
WinUsb_ResetDevice_t, 829
WinUsb_ResetPipe_t, 828
WinUsb_SetCurrentAlternateSetting_t, 829
WinUsb_SetPipePolicy_t, 829
WinUsb_SetPowerPolicy_t, 829
WinUsb_WritePipe_t, 829
winfd, 340
cancel_fn, 340
fd, 340
handle, 340
itransfer, 340
overlapped, 340

rw, 340
wint_t
stdint.h, 845
winusb_interface, 340
AbortPipe, 341
ControlTransfer, 341
FlushPipe, 341
Free, 341
GetAssociatedInterface, 341
GetCurrentAlternateSetting, 341
GetDescriptor, 341
GetOverlappedResult, 341
GetPipePolicy, 341
GetPowerPolicy, 341
Initialize, 341
initialized, 341
QueryDeviceInformation, 341
QueryInterfaceSettings, 341
QueryPipe, 341
ReadPipe, 341
ResetDevice, 341
ResetPipe, 341
SetCurrentAlternateSetting, 341
SetPipePolicy, 341
SetPowerPolicy, 341
WritePipe, 341
with_timing
AIOCommandLineOptions, 159
opts, 264
work
AIOContinuousBuf, 160
worker
AIOContinuousBuf, 160
workerBusy
AIOUSBDevice, 167
workerResult
AIOUSBDevice, 167
workerStatus
AIOUSBDevice, 167
wrappers, 23
write
AIOUSB::DigitalIOSubsystem, 223
AIOUSB::DIOStreamSubsystem, 228
write_sample.c
NO_DEVICE_FOUND, 877
SUCCESS, 877
USB_ERROR, 877
write_clock_rate
config_options, 199
write_sample.c
BITS_PER_BYTE, 876
DEVICES_REQUIRED, 876
EXIT_CODE, 876
find_dio_96, 877
fnd, 877
MASK_BYTES, 876
MAX_DIO_BYTES, 876
MAX_NAME_SIZE, 876
main, 877
writeBuffer
DeviceInfo, 216
TestCaseSetup, 274
writeConfig
AIOUSB::AnalogInputSubsystem, 174
WriteConfigBlock
AIOUSB_ADC.c, 559
AIOUSB_ADC.h, 592
libaiousb.h, 738
ndkbuild/jni/AIOUSB_ADC.c, 572
ndkbuild/jni/AIOUSB_ADC.h, 605
writeCounts
AIOUSB::AnalogOutputSubsystem, 191
WritePipe
winusb_interface, 341
writeValues
AIOUSB::DigitalIOSubsystem, 224
writeVolts
AIOUSB::AO16_AnalogOutputSubsystem, 194
AIOUSB::DA12_AnalogOutputSubsystem, 210

Xcode/config.h
DEFAULT_VISIBILITY, 755
ENABLE_LOGGING, 755
HAVE_GETTIMEOFDAY, 755
HAVE_POLL_H, 755
HAVE_SYS_TIME_H, 755
OS_DARWIN, 755
POLL_NFDS_TYPE, 755
THREADS_POSIX, 755
xusb.c
USE_GENERIC, 767
USE_HID, 767
USE_PS3, 767
USE_SCSI, 767
USE_XBOX, 767
xusb.c
B, 766
BOMS_GET_MAX_LUN, 767
BOMS_RESET, 767
be_to_int32, 766
bool, 766
CALL_CHECK, 766
ERR_EXIT, 766
false, 766
HID_GET_IDLE, 766
HID_GET_PROTOCOL, 766
HID_GET_REPORT, 766
HID_SET_IDLE, 766
HID_SET_PROTOCOL, 766
HID_SET_REPORT, 766
INQUIRY_LENGTH, 766
MAX_OS_FD_LENGTH, 767
main, 767
msleep, 766
RETRY_MAX, 766
test_type, 767
true, 766
usb_interface, 766

YAML
AIOUSB_Properties.h, 655
ndkbuild/jni/AIOUSB_Properties.h, 657