NAME

libusb20 - USB access library

LIBRARY

USB access library (libusb -lusb)

SYNOPSIS

#include <libusb20.h>

int
libusb20_tr_close(*struct libusb20_transfer *xfer*);

int
libusb20_tr_open(*struct libusb20_transfer *xfer*, *uint32_t max_buf_size*, *uint32_t max_frame_count*, *uint8_t ep_no*);

libusb20_tr_open_stream(*struct libusb20_transfer *xfer*, *uint32_t max_buf_size*, *uint32_t max_frame_count*, *uint8_t ep_no*, *uint16_t stream_id*);

*struct libusb20_transfer**
libusb20_tr_get_pointer(*struct libusb20_device *pdev*, *uint16_t tr_index*);

uint16_t
libusb20_tr_get_time_complete(*struct libusb20_transfer *xfer*);

uint32_t
libusb20_tr_get_actual_frames(*struct libusb20_transfer *xfer*);

uint32_t
libusb20_tr_get_actual_length(*struct libusb20_transfer *xfer*);

uint32_t
libusb20_tr_get_max_frames(*struct libusb20_transfer *xfer*);

uint32_t
libusb20_tr_get_max_packet_length(*struct libusb20_transfer *xfer*);

uint32_t
libusb20_tr_get_max_total_length(*struct libusb20_transfer *xfer*);

uint8_t
libusb20_tr_get_status(*struct libusb20_transfer *xfer*);

uint8_t
libusb20_tr_pending(*struct libusb20_transfer *xfer*);

void
libusb20_tr_callback_wrapper(*struct libusb20_transfer *xfer*);

void
libusb20_tr_clear_stall_sync(*struct libusb20_transfer *xfer*);

void
libusb20_tr_drain(*struct libusb20_transfer *xfer*);

void
libusb20_tr_set_buffer(*struct libusb20_transfer *xfer, void *buffer, uint16_t fr_index*);

void
libusb20_tr_set_callback(*struct libusb20_transfer *xfer, libusb20_tr_callback_t *cb*);

void
libusb20_tr_set_flags(*struct libusb20_transfer *xfer, uint8_t flags*);

uint32_t
libusb20_tr_get_length(*struct libusb20_transfer *xfer, uint16_t fr_index*);

void
libusb20_tr_set_length(*struct libusb20_transfer *xfer, uint32_t length, uint16_t fr_index*);

void
libusb20_tr_set_priv_sc0(*struct libusb20_transfer *xfer, void *sc0*);

void
libusb20_tr_set_priv_sc1(*struct libusb20_transfer *xfer, void *sc1*);

void
libusb20_tr_set_timeout(*struct libusb20_transfer *xfer, uint32_t timeout*);

void

libusb20_tr_set_total_frames(*struct libusb20_transfer *xfer, uint32_t nframes*);

void
libusb20_tr_setup_bulk(*struct libusb20_transfer *xfer, void *pbuf, uint32_t length, uint32_t timeout*);

void
libusb20_tr_setup_control(*struct libusb20_transfer *xfer, void *psetup, void *pbuf, uint32_t timeout*);

void
libusb20_tr_setup_intr(*struct libusb20_transfer *xfer, void *pbuf, uint32_t length, uint32_t timeout*);

void
libusb20_tr_setup_isoc(*struct libusb20_transfer *xfer, void *pbuf, uint32_t length, uint61_t fr_index*);

uint8_t
libusb20_tr_bulk_intr_sync(*struct libusb20_transfer *xfer, void *pbuf, uint32_t length,*
*uint32_t *pactlen, uint32_t timeout*);

void
libusb20_tr_start(*struct libusb20_transfer *xfer*);

void
libusb20_tr_stop(*struct libusb20_transfer *xfer*);

void
libusb20_tr_submit(*struct libusb20_transfer *xfer*);

*void **
libusb20_tr_get_priv_sc0(*struct libusb20_transfer *xfer*);

*void **
libusb20_tr_get_priv_sc1(*struct libusb20_transfer *xfer*);

*const char **
libusb20_dev_get_backend_name(*struct libusb20_device **);

int
libusb20_dev_get_port_path(*struct libusb20_device *pdev, uint8_t *buf, uint8_t bufsize*);

int

libusb20_dev_get_info(*struct libusb20_device *pdev, struct usb_device_info *pinfo*);

int
libusb20_dev_get_iface_desc(*struct libusb20_device *pdev, uint8_t iface_index, char *buf, uint8_t len*);

*const char **
libusb20_dev_get_desc(*struct libusb20_device *pdev*);

int
libusb20_dev_get_stats(*struct libusb20_device *pdev, struct libusb20_device_stats *pstats*);

int
libusb20_dev_close(*struct libusb20_device *pdev*);

int
libusb20_dev_detach_kernel_driver(*struct libusb20_device *pdev, uint8_t iface_index*);

int
libusb20_dev_set_config_index(*struct libusb20_device *pdev, uint8_t configIndex*);

int
libusb20_dev_get_debug(*struct libusb20_device *pdev*);

int
libusb20_dev_get_fd(*struct libusb20_device *pdev*);

int
libusb20_dev_kernel_driver_active(*struct libusb20_device *pdev, uint8_t iface_index*);

int
libusb20_dev_open(*struct libusb20_device *pdev, uint16_t transfer_max*);

int
libusb20_dev_process(*struct libusb20_device *pdev*);

int
libusb20_dev_request_sync(*struct libusb20_device *pdev,*
*struct LIBUSB20_CONTROL_SETUP_DECODED *setup, void *data, uint16_t *pactlen,*
uint32_t timeout, uint8_t flags);

int
libusb20_dev_req_string_sync(*struct libusb20_device *pdev, uint8_t index, uint16_t langid, void *ptr, uint16_t len*);

int
libusb20_dev_req_string_simple_sync(*struct libusb20_device *pdev, uint8_t index, void *ptr, uint16_t len*);

int
libusb20_dev_reset(*struct libusb20_device *pdev*);

int
libusb20_dev_check_connected(*struct libusb20_device *pdev*);

int
libusb20_dev_set_power_mode(*struct libusb20_device *pdev, uint8_t power_mode*);

uint8_t
libusb20_dev_get_power_mode(*struct libusb20_device *pdev*);

uint16_t
libusb20_dev_get_power_usage(*struct libusb20_device *pdev*);

int
libusb20_dev_set_alt_index(*struct libusb20_device *pdev, uint8_t iface_index, uint8_t alt_index*);

*struct LIBUSB20_DEVICE_DESC_DECODED **
libusb20_dev_get_device_desc(*struct libusb20_device *pdev*);

*struct libusb20_config **
libusb20_dev_alloc_config(*struct libusb20_device *pdev, uint8_t config_index*);

*struct libusb20_device **
libusb20_dev_alloc(*void*);

uint8_t
libusb20_dev_get_address(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_parent_address(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_parent_port(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_bus_number(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_mode(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_speed(*struct libusb20_device *pdev*);

uint8_t
libusb20_dev_get_config_index(*struct libusb20_device *pdev*);

void
libusb20_dev_free(*struct libusb20_device *pdev*);

void
libusb20_dev_set_debug(*struct libusb20_device *pdev, int debug*);

void
libusb20_dev_wait_process(*struct libusb20_device *pdev, int timeout*);

int
libusb20_be_get_template(*struct libusb20_backend *pbe, int *ptemp*);

int
libusb20_be_set_template(*struct libusb20_backend *pbe, int temp*);

int
libusb20_be_get_dev_quirk(*struct libusb20_backend *pber, uint16_t index, struct libusb20_quirk *pq*);

int
libusb20_be_get_quirk_name(*struct libusb20_backend *pbe, uint16_t index, struct libusb20_quirk *pq*);

int
libusb20_be_add_dev_quirk(*struct libusb20_backend *pbe, struct libusb20_quirk *pq*);

int

libusb20_be_remove_dev_quirk(*struct libusb20_backend *pbe, struct libusb20_quirk *pq*);

*struct libusb20_backend **
libusb20_be_alloc_default(*void*);

*struct libusb20_backend **
libusb20_be_alloc_freebsd(*void*);

*struct libusb20_backend **
libusb20_be_alloc_linux(*void*);

*struct libusb20_device **
libusb20_be_device_foreach(*struct libusb20_backend *pbe, struct libusb20_device *pdev*);

void
libusb20_be_dequeue_device(*struct libusb20_backend *pbe, struct libusb20_device *pdev*);

void
libusb20_be_enqueue_device(*struct libusb20_backend *pbe, struct libusb20_device *pdev*);

void
libusb20_be_free(*struct libusb20_backend *pbe*);

uint8_t
libusb20_me_get_1(*const struct libusb20_me_struct *me, uint16_t off*);

uint16_t
libusb20_me_get_2(*const struct libusb20_me_struct *me, uint16_t off*);

uint16_t
libusb20_me_encode(*void *pdata, uint16_t len, const void *pdecoded*);

uint16_t
libusb20_me_decode(*const void *pdata, uint16_t len, void *pdecoded*);

*const uint8_t **
libusb20_desc_foreach(*const struct libusb20_me_struct *me, const uint8_t *pdesc*);

*const char **
libusb20_strerror(*int code*);

*const char **
libusb20_error_name(*int code*);

DESCRIPTION

The **libusb20** library implements functions to be able to easily access and control USB through the USB file system interface. The **libusb20** interfaces are specific to the FreeBSD usb stack and are not available on other operating systems, portable applications should consider using libusb(3).

USB TRANSFER OPERATIONS

libusb20_tr_close() will release all kernel resources associated with an USB *xfer*. This function returns zero upon success. Non-zero return values indicate a LIBUSB20_ERROR value.

libusb20_tr_open() will allocate kernel buffer resources according to *max_buf_size* and *max_frame_count* associated with an USB *pxfer* and bind the transfer to the specified *ep_no*. *max_buf_size* is the minimum buffer size which the data transport layer has to support. If *max_buf_size* is zero, the **libusb20** library will use wMaxPacketSize to compute the buffer size. This can be useful for isochronous transfers. The actual buffer size can be greater than *max_buf_size* and is returned by **libusb20_tr_get_max_total_length**(). If *max_frame_count* is OR'ed with LIBUSB20_MAX_FRAME_PRE_SCALE the remaining part of the argument is converted from milliseconds into the actual number of frames rounded up, when this function returns. This flag is only valid for ISOCHRONOUS transfers and has no effect for other transfer types. The actual number of frames setup is found by calling **libusb20_tr_get_max_frames**(). This function returns zero upon success. Non-zero return values indicate a LIBUSB20_ERROR value.

libusb20_tr_open_stream() is identical to **libusb20_tr_open**() except that a stream ID can be specified for BULK endpoints having such a feature. **libusb20_tr_open**() can be used to open stream ID zero.

libusb20_tr_get_pointer() will return a pointer to the allocated USB transfer according to the *pdev* and *tr_index* arguments. This function returns NULL in case of failure.

libusb20_tr_get_time_complete() will return the completion time of an USB transfer in millisecond units. This function is most useful for isochronous USB transfers when doing echo cancelling.

libusb20_tr_get_actual_frames() will return the actual number of USB frames after an USB transfer completed. A value of zero means that no data was transferred.

libusb20_tr_get_actual_length() will return the sum of the actual length for all transferred USB frames for the given USB transfer.

libusb20_tr_get_max_frames() will return the maximum number of USB frames that were allocated

when an USB transfer was setup for the given USB transfer.

libusb20_tr_get_max_packet_length() will return the maximum packet length in bytes associated with the given USB transfer. The packet length can be used round up buffer sizes so that short USB packets are avoided for proxy buffers.

libusb20_tr_get_max_total_length() will return the maximum value for the data length sum of all USB frames associated with an USB transfer. In case of control transfers the value returned does not include the length of the SETUP packet, 8 bytes, which is part of frame zero. The returned value of this function is always aligned to the maximum packet size, wMaxPacketSize, of the endpoint which the USB transfer is bound to.

libusb20_tr_get_status() will return the status of an USB transfer. Status values are defined by a set of LIBUSB20_TRANSFER_XXX enums.

libusb20_tr_pending() will return non-zero if the given USB transfer is pending for completion. Else this function returns zero.

libusb20_tr_callback_wrapper() This is an internal function used to wrap asynchronous USB callbacks.

libusb20_tr_clear_stall_sync() This is an internal function used to synchronously clear the stall on the given USB transfer. Please see the USB specification for more information on stall clearing. If the given USB transfer is pending when this function is called, the USB transfer will complete with an error after that this function has been called.

libusb20_tr_drain() will stop the given USB transfer and will not return until the USB transfer has been stopped in hardware.

libusb20_tr_set_buffer() is used to set the *buffer* pointer for the given USB transfer and *fr_index*. Typically the frame index is zero.

libusb20_tr_set_callback() is used to set the USB callback for asynchronous USB transfers. The callback type is defined by libusb20_tr_callback_t.

libusb20_tr_set_flags() is used to set various USB flags for the given USB transfer.

LIBUSB20_TRANSFER_SINGLE_SHORT_NOT_OK Report a short frame as error.

LIBUSB20_TRANSFER_MULTI_SHORT_NOT_OK Multiple short frames are not allowed.

LIBUSB20_TRANSFER_FORCE_SHORT	All transmitted frames are short terminated.
LIBUSB20_TRANSFER_DO_CLEAR_STALL	Will do a clear-stall before starting the transfer.

libusb20_tr_get_length() returns the length of the given USB frame by index. After an USB transfer is complete the USB frame length will get updated to the actual transferred length.

libusb20_tr_set_length() sets the length of the given USB frame by index.

libusb20_tr_set_priv_sc0() sets private driver pointer number zero.

libusb20_tr_set_priv_sc1() sets private driver pointer number one.

libusb20_tr_set_timeout() sets the timeout for the given USB transfer. A timeout value of zero means no timeout. The timeout is given in milliseconds.

libusb20_tr_set_total_frames() sets the total number of frames that should be executed when the USB transfer is submitted. The total number of USB frames must be less than the maximum number of USB frames associated with the given USB transfer.

libusb20_tr_setup_bulk() is a helper function for setting up a single frame USB BULK transfer.

libusb20_tr_setup_control() is a helper function for setting up a single or dual frame USB CONTROL transfer depending on the control transfer length.

libusb20_tr_setup_intr() is a helper function for setting up a single frame USB INTERRUPT transfer.

libusb20_tr_setup_isoc() is a helper function for setting up a multi frame USB ISOCHRONOUS transfer.

libusb20_tr_bulk_intr_sync() will perform a synchronous BULK or INTERRUPT transfer having length given by the *length* argument and buffer pointer given by the *pbuf* argument on the USB transfer given by the *xfer* argument. If the *pactlen* argument is non-NULL the actual transfer length will be stored at the given pointer destination. If the *timeout* argument is non-zero the transfer will timeout after the given value in milliseconds. This function does not change the transfer flags, like short packet not ok. This function returns zero on success else a LIBUSB20_TRANSFER_XXX value is returned.

libusb20_tr_start() will get the USB transfer started, if not already started. This function will not get the transfer queued in hardware. This function is non-blocking.

libusb20_tr_stop() will get the USB transfer stopped, if not already stopped. This function is non-blocking, which means that the actual stop can happen after the return of this function.

libusb20_tr_submit() will get the USB transfer queued in hardware.

libusb20_tr_get_priv_sc0() returns private driver pointer number zero associated with an USB transfer.

libusb20_tr_get_priv_sc1() returns private driver pointer number one associated with an USB transfer.

USB DEVICE OPERATIONS

libusb20_dev_get_backend_name() returns a zero terminated string describing the backend used.

libusb20_dev_get_port_path() retrieves the list of USB port numbers which the datastream for a given USB device follows. The first port number is the Root HUB port number. Then children port numbers follow. The Root HUB device itself has a port path length of zero. Valid port numbers start at one and range until and including 255. Typically there should not be more than 16 levels, due to electrical and protocol limitations. This functions returns the number of actual port levels upon success else a LIBUSB20_ERROR value is returned which are always negative. If the actual number of port levels is greater than the maximum specified, a LIBUSB20_ERROR value is returned.

libusb20_dev_get_info() retrieves the BSD specific usb_device_info structure into the memory location given by *pinfo*. The USB device given by *pdev* must be opened before this function will succeed. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_get_iface_desc() retrieves the kernel interface description for the given USB *iface_index*. The format of the USB interface description is: "drivername<unit>: <description>" The description string is always zero terminated. A zero length string is written in case no driver is attached to the given interface. The USB device given by *pdev* must be opened before this function will succeed. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_get_desc() returns a zero terminated string describing the given USB device. The format of the string is: "drivername<unit>: <description>"

libusb20_dev_get_stats() retrieves the device statistics into the structure pointed to by the *pstats* argument. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_close() will close the given USB device. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_detach_kernel_driver() will try to detach the kernel driver for the USB interface given by

iface_index. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_set_config_index() will try to set the configuration index on an USB device. The first configuration index is zero. The un-configure index is 255. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_get_debug() returns the debug level of an USB device.

libusb20_dev_get_fd() returns the file descriptor of the given USB device. A negative value is returned when no file descriptor is present. The file descriptor can be used for polling purposes.

libusb20_dev_kernel_driver_active() returns zero if a kernel driver is active on the given USB interface. Else a LIBUSB20_ERROR value is returned.

libusb20_dev_open() opens an USB device so that setting up USB transfers becomes possible. The number of USB transfers can be zero which means only control transfers are allowed. This function returns zero on success else a LIBUSB20_ERROR value is returned. A return value of LIBUSB20_ERROR_BUSY means that the device is already opened.

libusb20_dev_process() is called to sync kernel USB transfers with userland USB transfers. This function returns zero on success else a LIBUSB20_ERROR value is returned typically indicating that the given USB device has been detached.

libusb20_dev_request_sync() will perform a synchronous control request on the given USB device. Before this call will succeed the USB device must be opened. *setup* is a pointer to a decoded and host endian SETUP packet. *data* is a pointer to a data transfer buffer associated with the control transaction. This argument can be NULL. *pactlen* is a pointer to a variable that will hold the actual transfer length after the control transaction is complete. *timeout* is the transaction timeout given in milliseconds. A timeout of zero means no timeout. *flags* is used to specify transaction flags, for example LIBUSB20_TRANSFER_SINGLE_SHORT_NOT_OK. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_req_string_sync() will synchronously request an USB string by language ID and string index into the given buffer limited by a maximum length. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_req_string_simple_sync() will synchronously request an USB string using the default language ID and convert the string into ASCII before storing the string into the given buffer limited by a maximum length which includes the terminating zero. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_reset() will try to BUS reset the given USB device and restore the last set USB configuration. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_check_connected() will check if an opened USB device is still connected. This function returns zero if the device is still connected else a LIBUSB20_ERROR value is returned.

libusb20_dev_set_power_mode() sets the power mode of the USB device. Valid power modes:

LIBUSB20_POWER_OFF

LIBUSB20_POWER_ON

LIBUSB20_POWER_SAVE

LIBUSB20_POWER_SUSPEND

LIBUSB20_POWER_RESUME

This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_get_power_mode() returns the currently selected power mode for the given USB device.

libusb20_dev_get_power_usage() returns the reported power usage in milliamps for the given USB device. A power usage of zero typically means that the device is self powered.

libusb20_dev_set_alt_index() will try to set the given alternate index for the given USB interface index. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_dev_get_device_desc() returns a pointer to the decoded and host endian version of the device descriptor. The USB device need not be opened when calling this function.

libusb20_dev_alloc_config() will read out and decode the USB config descriptor for the given USB device and config index. This function returns a pointer to the decoded configuration which must eventually be passed to free(). NULL is returned in case of failure.

libusb20_dev_alloc() is an internal function to allocate a new USB device.

libusb20_dev_get_address() returns the internal and not necessarily the real hardware address of the given USB device. Valid addresses start at one.

libusb20_dev_get_parent_address() returns the internal and not necessarily the real hardware address of the given parent USB HUB device. This value is zero for the root HUB which usually has a device address equal to one. Valid addresses start at one.

libusb20_dev_get_parent_port() returns the port number on the parent USB HUB device. This value is zero for the root HUB which usually has a device address equal to one. Valid port numbers start at one.

libusb20_dev_get_bus_number() returns the internal bus number which the given USB device belongs to. Valid bus numbers start at zero.

libusb20_dev_get_mode() returns the current operation mode of the USB entity. Valid return values are:

LIBUSB20_MODE_HOST

LIBUSB20_MODE_DEVICE

libusb20_dev_get_speed() returns the current speed of the given USB device.

LIBUSB20_SPEED_UNKNOWN

LIBUSB20_SPEED_LOW

LIBUSB20_SPEED_FULL

LIBUSB20_SPEED_HIGH

LIBUSB20_SPEED_VARIABLE

LIBUSB20_SPEED_SUPER

libusb20_dev_get_config_index() returns the currently selected config index for the given USB device.

libusb20_dev_free() will free the given USB device and all associated USB transfers.

libusb20_dev_set_debug() will set the debug level for the given USB device.

libusb20_dev_wait_process() will wait until a pending USB transfer has completed on the given USB device. A timeout value can be specified which is passed on to the poll(2) function.

USB BACKEND OPERATIONS

libusb20_be_get_template() will return the currently selected global USB device side mode template into the integer pointer *ptemp*. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_be_set_template() will set the global USB device side mode template to *temp*. The new template is not activated until after the next USB enumeration. The template number decides how the USB device will present itself to the USB Host, like Mass Storage Device, USB Ethernet Device. Also see the usb2_template(4) module. This function returns zero on success else a LIBUSB20_ERROR value is returned.

libusb20_be_get_dev_quirk() will return the device quirk according to *index* into the libusb20_quirk structure pointed to by *pq*. This function returns zero on success else a LIBUSB20_ERROR value is returned. If the given quirk does not exist LIBUSB20_ERROR_NOT_FOUND is returned.

libusb20_be_get_quirk_name() will return the quirk name according to *index* into the libusb20_quirk structure pointed to by *pq*. This function returns zero on success else a LIBUSB20_ERROR value is returned. If the given quirk does not exist LIBUSB20_ERROR_NOT_FOUND is returned.

libusb20_be_add_dev_quirk() will add the libusb20_quirk structure pointed to by the *pq* argument into the device quirk list. This function returns zero on success else a LIBUSB20_ERROR value is returned. If the given quirk cannot be added LIBUSB20_ERROR_NO_MEM is returned.

libusb20_be_remove_dev_quirk() will remove the quirk matching the libusb20_quirk structure pointed to by the *pq* argument from the device quirk list. This function returns zero on success else a LIBUSB20_ERROR value is returned. If the given quirk does not exist LIBUSB20_ERROR_NOT_FOUND is returned.

libusb20_be_alloc_default() **libusb20_be_alloc_freebsd**() **libusb20_be_alloc_linux**() These functions are used to allocate a specific USB backend or the operating system default USB backend. Allocating a backend is a way to scan for currently present USB devices.

libusb20_be_device_foreach() is used to iterate USB devices present in a USB backend. The starting value of *pdev* is NULL. This function returns the next USB device in the list. If NULL is returned the end of the USB device list has been reached.

libusb20_be_dequeue_device() will dequeue the given USB device pointer from the backend USB device list. Dequeued USB devices will not be freed when the backend is freed.

libusb20_be_enqueue_device() will enqueue the given USB device pointer in the backend USB device list. Enqueued USB devices will get freed when the backend is freed.

libusb20_be_free() will free the given backend and all USB devices in its device list.

USB DESCRIPTOR PARSING

libusb20_me_get_1(*pie*, *offset*) This function will return a byte at the given byte offset of a message entity. This function is safe against invalid offsets.

libusb20_me_get_2(*pie*, *offset*) This function will return a little endian 16-bit value at the given byte offset of a message entity. This function is safe against invalid offsets.

libusb20_me_encode(*pbuf*, *len*, *pdecoded*) This function will encode a so-called *DECODED structure into binary format. The total encoded length that will fit in the given buffer is returned. If the buffer pointer is NULL no data will be written to the buffer location.

libusb20_me_decode(*pbuf*, *len*, *pdecoded*) This function will decode a binary structure into a so-called *DECODED structure. The total decoded length is returned. The buffer pointer cannot be NULL.

USB DEBUGGING

const char * **libusb20_strerror**(*int code*) Get the ASCII representation of the error given by the *code* argument. This function does not return NULL.

const char * **libusb20_error_name**(*int code*) Get the ASCII representation of the error enum given by the *code* argument. This function does not return NULL.

FILES

/dev/usb

SEE ALSO

libusb(3), usb(4), usbconfig(8), usbdump(8)

HISTORY

Some parts of the **libusb20** API derives from the libusb project at sourceforge.