

**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, uint64\_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 *paclen* 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. *packetlen* 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.