

**NAME**

**libusb** - USB access library

**LIBRARY**

USB access library (libusb, -lusb)

**SYNOPSIS**

```
#include <libusb.h>
```

**DESCRIPTION**

The **libusb** library contains interfaces for directly managing a usb device. The current implementation supports v1.0 of the libusb API.

**LIBRARY INITIALISATION AND DEINITIALISATION**

*const struct libusb\_version \* libusb\_get\_version(void)* This function returns version information about LibUSB.

*int libusb\_init(libusb\_context \*\*ctx)* Call this function before any other libusb v1.0 API function, to initialise a valid libusb v1.0 context. If the *ctx* argument is non-NULL, a pointer to the libusb context is stored at the given location. This function returns 0 upon success or LIBUSB\_ERROR on failure.

*int libusb\_init\_context(libusb\_context \*\*ctx, const struct libusb\_init\_option [], int num\_options)* Call this function before any other libusb v1.0 API function, to initialise a valid libusb v1.0 context. If the *ctx* argument is non-NULL, a pointer to the libusb context is stored at the given location. Additional options, like the USB debug level, may be given using the second and third argument. If no options are needed, simply use libusb\_init(). This function returns 0 upon success or a LIBUSB\_ERROR value on failure.

*void libusb\_exit(libusb\_context \*ctx)* Deinitialise libusb. Must be called at the end of the application. Other libusb routines may not be called after this function.

*int libusb\_has\_capability(uint32\_t capability)* This function checks the runtime capabilities of **libusb**. This function will return non-zero if the given *capability* is supported, 0 if it is not supported. The valid values for *capability* are:

*LIBUSB\_CAP\_HAS\_CAPABILITY*

**libusb** supports **libusb\_has\_capability()**.

*LIBUSB\_CAP\_HAS\_HOTPLUG*

**libusb** supports hotplug notifications.

**LIBUSB\_CAP\_HAS\_HID\_ACCESS**

**libusb** can access HID devices without requiring user intervention.

**LIBUSB\_CAP\_SUPPORTS\_DETACH\_KERNEL\_DRIVER**

**libusb** supports detaching of the default USB driver with **libusb\_detach\_kernel\_driver()**.

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

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

*void* **libusb\_set\_debug**(*libusb\_context \*ctx, int level*) Set the debug level to *level*.

*ssize\_t* **libusb\_get\_device\_list**(*libusb\_context \*ctx, libusb\_device \*\*\*list*) Populate *list* with the list of usb devices available, adding a reference to each device in the list. All the list entries created by this function must have their reference counter decremented when you are done with them, and the list itself must be freed. This function returns the number of devices in the list or a LIBUSB\_ERROR code.

*void* **libusb\_free\_device\_list**(*libusb\_device \*\*list, int unref\_devices*) Free the list of devices discovered by **libusb\_get\_device\_list**. If *unref\_device* is set to 1 all devices in the list have their reference counter decremented once.

*uint8\_t* **libusb\_get\_bus\_number**(*libusb\_device \*dev*) Returns the number of the bus contained by the device *dev*.

*uint8\_t* **libusb\_get\_port\_number**(*libusb\_device \*dev*) Returns the port number which the device given by *dev* is attached to.

*int* **libusb\_get\_port\_numbers**(*libusb\_device \*dev, uint8\_t \*buf, uint8\_t bufsize*) Stores, in the buffer *buf* of size *bufsize*, the list of all port numbers from root for the device *dev*.

*int* **libusb\_get\_port\_path**(*libusb\_context \*ctx, libusb\_device \*dev, uint8\_t \*buf, uint8\_t bufsize*)  
Deprecated function equivalent to **libusb\_get\_port\_numbers**.

*uint8\_t* **libusb\_get\_device\_address**(*libusb\_device \*dev*) Returns the *device\_address* contained by the device *dev*.

*enum libusb\_speed* **libusb\_get\_device\_speed**(*libusb\_device \*dev*) Returns the wire speed at which the

device is connected. See the LIBUSB\_SPEED\_XXX enums for more information. LIBUSB\_SPEED\_UNKNOWN is returned in case of unknown wire speed.

*int libusb\_get\_max\_packet\_size(libusb\_device \*dev, unsigned char endpoint)* Returns the wMaxPacketSize value on success, LIBUSB\_ERROR\_NOT\_FOUND if the endpoint does not exist and LIBUSB\_ERROR\_OTHERS on other failure.

*int libusb\_get\_max\_iso\_packet\_size(libusb\_device \*dev, unsigned char endpoint)* Returns the packet size multiplied by the packet multiplier on success, LIBUSB\_ERROR\_NOT\_FOUND if the endpoint does not exist and LIBUSB\_ERROR\_OTHERS on other failure.

*libusb\_device \* libusb\_ref\_device(libusb\_device \*dev)* Increment the reference counter of the device *dev*.

*void libusb\_unref\_device(libusb\_device \*dev)* Decrement the reference counter of the device *dev*.

*int libusb\_open(libusb\_device \*dev, libusb\_device\_handle \*\*devh)* Open a device and obtain a device\_handle. Returns 0 on success, LIBUSB\_ERROR\_NO\_MEM on memory allocation problems, LIBUSB\_ERROR\_ACCESS on permissions problems, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on other errors.

*libusb\_device\_handle \* libusb\_open\_device\_with\_vid\_pid(libusb\_context \*ctx, uint16\_t vid, uint16\_t pid)* A convenience function to open a device by vendor and product IDs *vid* and *pid*. Returns NULL on error.

*void libusb\_close(libusb\_device\_handle \*devh)* Close a device handle.

*libusb\_device \* libusb\_get\_device(libusb\_device\_handle \*devh)* Get the device contained by *devh*. Returns NULL on error.

*int libusb\_get\_configuration(libusb\_device\_handle \*devh, int \*config)* Returns the value of the current configuration. Returns 0 on success, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on error.

*int libusb\_set\_configuration(libusb\_device\_handle \*devh, int config)* Set the active configuration to *config* for the device contained by *devh*. This function returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the requested configuration does not exist, LIBUSB\_ERROR\_BUSY if the interfaces are currently claimed, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_claim\_interface(libusb\_device\_handle \*devh, int interface\_number)* Claim an interface in a given *libusb\_device\_handle devh*. This is a non-blocking function. It returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the requested interface does not exist, LIBUSB\_ERROR\_BUSY if a program or driver has claimed the interface, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_release\_interface(libusb\_device\_handle \*devh, int interface\_number)* This function releases an interface. All the claimed interfaces on a device must be released before closing the device. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the interface was not claimed, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and LIBUSB\_ERROR on failure.

*int libusb\_set\_interface\_alt\_setting(libusb\_device\_handle \*dev, int interface\_number, int alternate\_setting)* Activate an alternate setting for an interface. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the interface was not claimed or the requested setting does not exist, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_clear\_halt(libusb\_device\_handle \*devh, unsigned char endpoint)* Clear an halt/stall for a endpoint. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the endpoint does not exist, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_reset\_device(libusb\_device\_handle \*devh)* Perform an USB port reset for an usb device. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if re-enumeration is required or if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_check\_connected(libusb\_device\_handle \*devh)* Test if the USB device is still connected. Returns 0 on success, LIBUSB\_ERROR\_NO\_DEVICE if it has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_kernel\_driver\_active(libusb\_device\_handle \*devh, int interface)* Determine if a driver is active on a interface. Returns 0 if no kernel driver is active and 1 if a kernel driver is active, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on failure.

*int libusb\_get\_driver(libusb\_device\_handle \*devh, int interface, char \*name, int namelen)* or *int libusb\_get\_driver\_np(libusb\_device\_handle \*devh, int interface, char \*name, int namelen)* Copy the name of the driver attached to the given *device* and *interface* into the buffer *name* of length *namelen*. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if no kernel driver is attached to the given interface and LIBUSB\_ERROR\_INVALID\_PARAM if the interface does not exist. This function is

non-portable. The buffer pointed to by *name* is only zero terminated on success.

*int libusb\_detach\_kernel\_driver*(*libusb\_device\_handle \*devh, int interface*) or *int libusb\_detach\_kernel\_driver\_np*(*libusb\_device\_handle \*devh, int interface*) Detach a kernel driver from an interface. This is needed to claim an interface already claimed by a kernel driver. 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 and a LIBUSB\_ERROR code on failure. This function is non-portable.

*int libusb\_attach\_kernel\_driver*(*libusb\_device\_handle \*devh, int interface*) Re-attach an interface kernel driver that was previously detached. Returns 0 on success, LIBUSB\_ERROR\_INVALID\_PARAM if the interface does not exist, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected, LIBUSB\_ERROR\_BUSY if the driver cannot be attached because the interface is claimed by a program or driver and a LIBUSB\_ERROR code on failure.

*int libusb\_set\_auto\_detach\_kernel\_driver*(*libusb\_device\_handle \*devh, int enable*) This function enables automatic kernel interface driver detach when an interface is claimed. When the interface is restored the kernel driver is allowed to be re-attached. If the *enable* argument is non-zero the feature is enabled. Else disabled. Returns 0 on success and a LIBUSB\_ERROR code on failure.

## USB DESCRIPTORS

*int libusb\_get\_device\_descriptor*(*libusb\_device \*dev, libusb\_device\_descriptor \*desc*) Get the USB device descriptor for the device *dev*. This is a non-blocking function. Returns 0 on success and a LIBUSB\_ERROR code on failure.

*int libusb\_get\_active\_config\_descriptor*(*libusb\_device \*dev, struct libusb\_config\_descriptor \*\*config*) Get the USB configuration descriptor for the active configuration. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the device is in an unconfigured state and a LIBUSB\_ERROR code on error.

*int libusb\_get\_config\_descriptor*(*libusb\_device \*dev, uint8\_t config\_index, libusb\_config\_descriptor \*\*config*) Get a USB configuration descriptor based on its index *idx*. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the configuration does not exist and a LIBUSB\_ERROR code on error.

*int libusb\_get\_config\_descriptor\_by\_value*(*libusb\_device \*dev, uint8\_t bConfigurationValue, libusb\_config\_descriptor \*\*config*) Get a USB configuration descriptor with a specific *bConfigurationValue*. This is a non-blocking function which does not send a request through the device. Returns 0 on success, LIBUSB\_ERROR\_NOT\_FOUND if the configuration does not exist and a

LIBUSB\_ERROR code on failure.

**void libusb\_free\_config\_descriptor**(*libusb\_config\_descriptor \*config*) Free a configuration descriptor.

**int libusb\_get\_string\_descriptor**(*libusb\_device\_handle \*devh, uint8\_t desc\_idx, uint16\_t langid, unsigned char \*data, int length*) Retrieve a string descriptor in raw format. Returns the number of bytes actually transferred on success or a negative LIBUSB\_ERROR code on failure.

**int libusb\_get\_string\_descriptor\_ascii**(*libusb\_device\_handle \*devh, uint8\_t desc\_idx, unsigned char \*data, int length*) Retrieve a string descriptor in C style ASCII. Returns the positive number of bytes in the resulting ASCII string on success and a LIBUSB\_ERROR code on failure.

**int libusb\_parse\_ss\_endpoint\_comp**(*const void \*buf, int len, libusb\_ss\_endpoint\_companion\_descriptor \*\*ep\_comp*) This function parses the USB 3.0 endpoint companion descriptor in host endian format pointed to by *buf* and having a length of *len*. Typically these arguments are the extra and extra\_length fields of the endpoint descriptor. On success the pointer to resulting descriptor is stored at the location given by *ep\_comp*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed USB 3.0 endpoint companion descriptor must be freed using the `libusb_free_ss_endpoint_comp` function.

**void libusb\_free\_ss\_endpoint\_comp**(*libusb\_ss\_endpoint\_companion\_descriptor \*ep\_comp*) This function is NULL safe and frees a parsed USB 3.0 endpoint companion descriptor given by *ep\_comp*.

**int libusb\_get\_ss\_endpoint\_companion\_descriptor**(*struct libusb\_context \*ctx, const struct libusb\_endpoint\_descriptor \*endpoint, struct libusb\_ss\_endpoint\_companion\_descriptor \*\*ep\_comp*) This function finds and parses the USB 3.0 endpoint companion descriptor given by *endpoint*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed USB 3.0 endpoint companion descriptor must be freed using the `libusb_free_ss_endpoint_companion_descriptor` function.

**void libusb\_free\_ss\_endpoint\_companion\_descriptor**(*struct libusb\_ss\_endpoint\_companion\_descriptor \*ep\_comp*) This function is NULL safe and frees a parsed USB 3.0 endpoint companion descriptor given by *ep\_comp*.

**int libusb\_get\_bos\_descriptor**(*libusb\_device\_handle \*handle, struct libusb\_bos\_descriptor \*\*bos*) This function queries the USB device given by *handle* and stores a pointer to a parsed BOS descriptor into *bos*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed BOS descriptor must be freed using the `libusb_free_bos_descriptor` function.

**int libusb\_parse\_bos\_descriptor**(*const void \*buf, int len, libusb\_bos\_descriptor \*\*bos*) This function parses a Binary Object Store, BOS, descriptor into host endian format pointed to by *buf* and having a

length of *len*. On success the pointer to resulting descriptor is stored at the location given by *bos*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed BOS descriptor must be freed using the `libusb_free_bos_descriptor` function.

**`void libusb_free_bos_descriptor`**(*libusb\_bos\_descriptor \*bos*) This function is NULL safe and frees a parsed BOS descriptor given by *bos*.

**`int 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*) This function parses the USB 2.0 extension descriptor from the descriptor given by *dev\_cap* and stores a pointer to the parsed descriptor into *usb\_2\_0\_extension*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed USB 2.0 extension descriptor must be freed using the `libusb_free_usb_2_0_extension_descriptor` function.

**`void libusb_free_usb_2_0_extension_descriptor`**(*struct libusb\_usb\_2\_0\_extension\_descriptor \*usb\_2\_0\_extension*) This function is NULL safe and frees a parsed USB 2.0 extension descriptor given by *usb\_2\_0\_extension*.

**`int 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\_capability*) This function parses the SuperSpeed device capability descriptor from the descriptor given by *dev\_cap* and stores a pointer to the parsed descriptor into *ss\_usb\_device\_capability*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed SuperSpeed device capability descriptor must be freed using the `libusb_free_ss_usb_device_capability_descriptor` function.

**`void libusb_free_ss_usb_device_capability_descriptor`**(*struct libusb\_ss\_usb\_device\_capability\_descriptor \*ss\_usb\_device\_capability*) This function is NULL safe and frees a parsed SuperSpeed device capability descriptor given by *ss\_usb\_device\_capability*.

**`int libusb_get_container_id_descriptor`**(*struct libusb\_context \*ctx, struct libusb\_bos\_dev\_capability\_descriptor \*dev\_cap, struct libusb\_container\_id\_descriptor \*\*container\_id*) This function parses the container ID descriptor from the descriptor given by *dev\_cap* and stores a pointer to the parsed descriptor into *container\_id*. Returns zero on success and a LIBUSB\_ERROR code on failure. On success the parsed container ID descriptor must be freed using the `libusb_free_container_id_descriptor` function.

**`void libusb_free_container_id_descriptor`**(*struct libusb\_container\_id\_descriptor \*container\_id*) This function is NULL safe and frees a parsed container ID descriptor given by *container\_id*.

## USB ASYNCHRONOUS I/O

*struct libusb\_transfer* \* **libusb\_alloc\_transfer**(*int iso\_packets*) Allocate a transfer with the number of isochronous packet descriptors specified by *iso\_packets*. Returns NULL on error.

*void* **libusb\_free\_transfer**(*struct libusb\_transfer* \**tr*) Free a transfer.

*int* **libusb\_submit\_transfer**(*struct libusb\_transfer* \**tr*) This function will submit a transfer and returns immediately. Returns 0 on success, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on other failure.

*int* **libusb\_cancel\_transfer**(*struct libusb\_transfer* \**tr*) This function asynchronously cancels a transfer. Returns 0 on success and a LIBUSB\_ERROR code on failure.

## USB SYNCHRONOUS I/O

*int* **libusb\_control\_transfer**(*libusb\_device\_handle* \**devh*, *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. Returns the actual number of bytes transferred on success, in the range from and including zero up to and including *wLength*. On error a LIBUSB\_ERROR code is returned, for example LIBUSB\_ERROR\_TIMEOUT if the transfer timed out, LIBUSB\_ERROR\_PIPE if the control request was not supported, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and another LIBUSB\_ERROR code on other failures. The LIBUSB\_ERROR codes are all negative.

*int* **libusb\_bulk\_transfer**(*struct libusb\_device\_handle* \**devh*, *unsigned char endpoint*, *unsigned char* \**data*, *int length*, *int* \**transferred*, *unsigned int timeout*) Perform an USB bulk transfer. A timeout value of zero means no timeout. The timeout value is given in milliseconds. Returns 0 on success, LIBUSB\_ERROR\_TIMEOUT if the transfer timed out, LIBUSB\_ERROR\_PIPE if the control request was not supported, LIBUSB\_ERROR\_OVERFLOW if the device offered more data, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on other failure.

*int* **libusb\_interrupt\_transfer**(*struct libusb\_device\_handle* \**devh*, *unsigned char endpoint*, *unsigned char* \**data*, *int length*, *int* \**transferred*, *unsigned int timeout*) Perform an USB Interrupt transfer. A timeout value of zero means no timeout. The timeout value is given in milliseconds. Returns 0 on success, LIBUSB\_ERROR\_TIMEOUT if the transfer timed out, LIBUSB\_ERROR\_PIPE if the control request was not supported, LIBUSB\_ERROR\_OVERFLOW if the device offered more data, LIBUSB\_ERROR\_NO\_DEVICE if the device has been disconnected and a LIBUSB\_ERROR code on other failure.

## USB STREAMS SUPPORT

*int* **libusb\_alloc\_streams**(*libusb\_device\_handle* \**dev*, *uint32\_t num\_streams*, *unsigned char* \**endpoints*,



*int num\_endpoints*) This function verifies that the given number of streams using the given number of endpoints is allowed and allocates the resources needed to use so-called USB streams. Currently only a single stream per endpoint is supported to simplify the internals of LibUSB. This function returns 0 on success or a LIBUSB\_ERROR code on failure.

*int libusb\_free\_streams(libusb\_device\_handle \*dev, unsigned char \*endpoints, int num\_endpoints)* This function release resources needed for streams usage. Returns 0 on success or a LIBUSB\_ERROR code on failure.

*void libusb\_transfer\_set\_stream\_id(struct libusb\_transfer \*transfer, uint32\_t stream\_id)* This function sets the stream ID for the given USB transfer.

*uint32\_t libusb\_transfer\_get\_stream\_id(struct libusb\_transfer \*transfer)* This function returns the stream ID for the given USB transfer. If no stream ID is used a value of zero is returned.

## USB EVENTS

*int libusb\_try\_lock\_events(libusb\_context \*ctx)* Try to acquire the event handling lock. Returns 0 if the lock was obtained and 1 if not.

*void libusb\_lock\_events(libusb\_context \*ctx)* Acquire the event handling lock. This function is blocking.

*void libusb\_unlock\_events(libusb\_context \*ctx)* Release the event handling lock. This will wake up any thread blocked on **libusb\_wait\_for\_event()**.

*int libusb\_event\_handling\_ok(libusb\_context \*ctx)* Determine if it still OK for this thread to be doing event handling. Returns 1 if event handling can start or continue. Returns 0 if this thread must give up the events lock.

*int libusb\_event\_handler\_active(libusb\_context \*ctx)* Determine if an active thread is handling events. Returns 1 if there is a thread handling events and 0 if there are no threads currently handling events.

*void libusb\_interrupt\_event\_handler(libusb\_context \*ctx)* Causes the **libusb\_handle\_events()** family of functions to return to the caller one time. The **libusb\_handle\_events()** functions may be called again after calling this function.

*void libusb\_lock\_event\_waiters(libusb\_context \*ctx)* Acquire the event\_waiters lock. This lock is designed to be obtained in 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.

*void libusb\_unlock\_event\_waiters(libusb\_context \*ctx)* Release the event\_waiters lock.

*int 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 will block until the timeout expires or a transfer completes or a thread releases the event handling lock through **libusb\_unlock\_events()**. Returns 0 after a transfer completes or another thread stops event handling, and 1 if the timeout expired.

*int libusb\_handle\_events\_timeout\_completed(libusb\_context \*ctx, struct timeval \*tv, int \*completed)* Handle any pending events by checking if timeouts have expired and by checking the set of file descriptors for activity. If the *completed* argument is not equal to NULL, this function will loop until a transfer completion callback sets the variable pointed to by the *completed* argument to non-zero. If the *tv* argument is not equal to NULL, this function will return LIBUSB\_ERROR\_TIMEOUT after the given timeout. Returns 0 on success, or a LIBUSB\_ERROR code on failure or timeout.

*int libusb\_handle\_events\_completed(libusb\_context \*ctx, int \*completed)* Handle any pending events by checking the set of file descriptors for activity. If the *completed* argument is not equal to NULL, this function will loop until a transfer completion callback sets the variable pointed to by the *completed* argument to non-zero. Returns 0 on success, or a LIBUSB\_ERROR code on failure.

*int libusb\_handle\_events\_timeout(libusb\_context \*ctx, struct timeval \*tv)* Handle any pending events by checking if timeouts have expired and by checking the set of file descriptors for activity. Returns 0 on success, or a LIBUSB\_ERROR code on failure or timeout.

*int libusb\_handle\_events(libusb\_context \*ctx)* Handle any pending events in blocking mode with a sensible timeout. Returns 0 on success and a LIBUSB\_ERROR code on failure.

*int libusb\_handle\_events\_locked(libusb\_context \*ctx, struct timeval \*tv)* Handle any pending events by polling file descriptors, without checking if another thread is already doing so. Must be called with the event lock held.

*int libusb\_get\_next\_timeout(libusb\_context \*ctx, struct timeval \*tv)* Determine the next internal timeout that libusb needs to handle. Returns 0 if there are no pending timeouts, 1 if a timeout was returned, or a LIBUSB\_ERROR code on failure or timeout.

*void libusb\_set\_pollfd\_notifiers(libusb\_context \*ctx, libusb\_pollfd\_added\_cb added\_cb, libusb\_pollfd\_removed\_cb remove\_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.

*const struct libusb\_pollfd \*\** **libusb\_get\_pollfds**(*libusb\_context \*ctx*) Retrieve a list of file descriptors that should be polled by your main loop as libusb event sources. Returns a NULL-terminated list on success or NULL on failure.

*int* **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 \*handle*) This function registers a hotplug filter. The *events* argument select which events makes the hotplug filter trigger. Available event values are LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_ARRIVED and LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_LEFT. One or more events must be specified. The *vendor\_id*, *product\_id* and *dev\_class* arguments can be set to LIBUSB\_HOTPLUG\_MATCH\_ANY to match any value in the USB device descriptor. Else the specified value is used for matching. If the *flags* argument is set to LIBUSB\_HOTPLUG\_ENUMERATE, all currently attached and matching USB devices will be passed to the hotplug filter, given by the *cb\_fn* argument. Else the *flags* argument should be set to LIBUSB\_HOTPLUG\_NO\_FLAGS. This function returns 0 upon success or a LIBUSB\_ERROR code on failure.

*int* **libusb\_hotplug\_callback\_fn**(*libusb\_context \*ctx, libusb\_device \*device, libusb\_hotplug\_event event, void \*user\_data*) The hotplug filter function. If this function returns non-zero, the filter is removed. Else the filter is kept and can receive more events. The *user\_data* argument is the same as given when the filter was registered. The *event* argument can be either of LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_ARRIVED or LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_LEFT.

*void* **libusb\_hotplug\_deregister\_callback**(*libusb\_context \*ctx, libusb\_hotplug\_callback\_handle handle*) This function unregisters a hotplug filter.

## LIBUSB VERSION 0.1 COMPATIBILITY

The library is also compliant with LibUSB version 0.1.12.

**usb\_open()** **usb\_close()** **usb\_get\_string()** **usb\_get\_string\_simple()** **usb\_get\_descriptor\_by\_endpoint()**  
**usb\_get\_descriptor()** **usb\_parse\_descriptor()** **usb\_parse\_configuration()** **usb\_destroy\_configuration()**  
**usb\_fetch\_and\_parse\_descriptors()** **usb\_bulk\_write()** **usb\_bulk\_read()** **usb\_interrupt\_write()**  
**usb\_interrupt\_read()** **usb\_control\_msg()** **usb\_set\_configuration()** **usb\_claim\_interface()**  
**usb\_release\_interface()** **usb\_set\_altinterface()** **usb\_resetep()** **usb\_clear\_halt()** **usb\_reset()** **usb\_strerror()**  
**usb\_init()** **usb\_set\_debug()** **usb\_find\_busses()** **usb\_find\_devices()** **usb\_device()** **usb\_get\_busses()**  
**usb\_check\_connected()** **usb\_get\_driver\_np()** **usb\_detach\_kernel\_driver\_np()**

## SEE ALSO

libusb20(3), usb(4), usbconfig(8), usbdump(8)

**<https://libusb.info/>**

## **HISTORY**

**libusb** support first appeared in FreeBSD 8.0.