NAME

usbhid, **hid_get_report_desc**, **hid_get_report_id**, **hid_use_report_desc**, **hid_dispose_report_desc**, **hid_start_parse**, **hid_end_parse**, **hid_get_item**, **hid_report_size**, **hid_locate**, **hid_usage_page**, **hid_usage_in_page**, **hid_init**, **hid_get_data**, **hid_set_data**, **hid_get_report**, **hid_set_report** - USB HID access routines

LIBRARY

USB Human Interface Devices Library (libusbhid, -lusbhid)

SYNOPSIS

#include <usbhid.h>

report_desc_t
hid_get_report_desc(*int file*);

int
hid_get_report_id(*int file*);

int
hid_set_immed(*int fd, int enable*);

report_desc_t
hid_use_report_desc(*unsigned char *data, unsigned int size*);

void
hid_dispose_report_desc(*report_desc_t d*);

hid_data_t
hid_start_parse(*report_desc_t d, int kindset, int id*);

void
hid_end_parse(*hid_data_t s*);

int
hid_get_item(*hid_data_t s, hid_item_t *h*);

int
hid_report_size(*report_desc_t d, hid_kind_t k, int id*);

int

hid_locate(*report_desc_t d, u_int usage, hid_kind_t k, hid_item_t *h, int id*);

*const char **
hid_usage_page(*int i*);

*const char **
hid_usage_in_page(*u_int u*);

int
hid_parse_usage_page(*const char **);

int
hid_parse_usage_in_page(*const char **);

void
hid_init(*const char *file*);

int
hid_get_data(*const void *data, const hid_item_t *h*);

void
hid_set_data(*void *buf, const hid_item_t *h, int data*);

int
hid_get_report(*int fd, enum hid_kind k, unsigned char *data, unsigned int size*);

int
hid_set_report(*int fd, enum hid_kind k, unsigned char *data, unsigned int size*);

DESCRIPTION

The **usbhid** library provides routines to extract data from USB Human Interface Devices.

Introduction

USB HID devices send and receive data laid out in a device dependent way. The **usbhid** library contains routines to extract the *report descriptor* which contains the data layout information and then use this information.

The routines can be divided into four parts: extraction of the descriptor, parsing of the descriptor, translating to/from symbolic names, and data manipulation.

Synchronous HID operation

Synchronous HID operation can be enabled or disabled by a call to **hid_set_immed**(). If the second argument is zero synchronous HID operation is disabled. Else synchronous HID operation is enabled. The function returns a negative value on failure.

hid_get_report() and **hid_set_report**() functions allow to synchronously get and set specific report if device supports it. For devices with multiple report IDs, wanted ID should be provided in the first byte of the buffer for both get and set.

Descriptor Functions

The report descriptor ID can be obtained by calling **hid_get_report_id**(). A report descriptor can be obtained by calling **hid_get_report_desc**() with a file descriptor obtained by opening a uhid(4) device. Alternatively a data buffer containing the report descriptor can be passed into **hid_use_report_desc**(). The data is copied into an internal structure. When the report descriptor is no longer needed it should be freed by calling **hid_dispose_report_desc**(). The type *report_desc_t* is opaque and should be used when calling the parsing functions. If **hid_dispose_report_desc**() fails it will return NULL.

Descriptor Parsing Functions

To parse the report descriptor the **hid_start_parse**() function should be called with a report descriptor, a set that describes which items that are interesting, and the desired report ID (or -1 to obtain items of all report IDs). The set is obtained by OR-ing together values *(1 << k)* where *k* is an item of type *hid_kind_t*. The function returns NULL if the initialization fails, otherwise an opaque value to be used in subsequent calls. After parsing the **hid_end_parse**() function should be called to free internal data structures.

To iterate through all the items in the report descriptor **hid_get_item**() should be called while it returns a value greater than 0. When the report descriptor ends it will returns 0; a syntax error within the report descriptor will cause a return value less than 0. The struct pointed to by *h* will be filled with the relevant data for the item. The definition of *hid_item_t* can be found in <*usbhid.h*> and the meaning of the components in the USB HID documentation.

Data should be read/written to the device in the size of the report. The size of a report (of a certain kind) can be computed by the **hid_report_size**() function. If the report is prefixed by an ID byte it is given by *id*.

To locate a single item the **hid_locate**() function can be used. It should be given the usage code of the item and its kind and it will fill the item and return non-zero if the item was found.

Name Translation Functions

The function **hid_usage_page**() will return the symbolic name of a usage page, and the function

hid_usage_in_page() will return the symbolic name of the usage within the page. Both these functions may return a pointer to static data.

The functions **hid_parse_usage_page**() and **hid_parse_usage_in_page**() are the inverses of **hid_usage_page**() and **hid_usage_in_page**(). They take a usage string and return the number of the usage, or -1 if it cannot be found.

Before any of these functions can be called the usage table must be parsed, this is done by calling **hid_init**() with the name of the table. Passing NULL to this function will cause it to use the default table.

Data Extraction Functions

Given the data obtained from a HID device and an item in the report descriptor the **hid_get_data**() function extracts the value of the item. Conversely **hid_set_data**() can be used to put data into a report (which must be zeroed first).

FILES

/usr/share/misc/usb_hid_usages The default HID usage table.

EXAMPLES

Not yet.

SEE ALSO

The USB specifications can be found at *http://www.usb.org/developers/docs/*.

uhid(4), usb(4)

HISTORY

The **usbhid** library first appeared in NetBSD 1.5.

BUGS

This man page is woefully incomplete.