Libxbee3 Manual

目錄

XBEE_INIT

NAME

xbee_init, xbee_fini

SYNOPSIS

#include <xbee.h>
void xbee_init(void);
void xbee_fini(void);

DESCRIPTION

The functions **xbee_init**() and **xbee_fini**() are used to manually setup and teardown libxbee. These functions should be used with care, and in conjunction with
the **XBEE_MANUAL_INIT** and **XBEE_MANUAL_FINI** compile options.
You must call **xbee_init**() once before calling on any other libxbee functionality.
Similarly, you must only call **xbee_fini**() after all instances of libxbee have been terminated.
Calling **xbee_init**() multiple times will result in undefined behavior.

XBEE_SETUP

NAME

xbee_setup, xbee_vsetup, xbee_validate, xbee_shutdown

SYNOPSIS

#include <xbee.h>
xbee_err xbee_setup(struct xbee *****retXbee*, const char ****mode***, ...);**
xbee_err xbee_vsetup(struct xbee *****retXbee*, const char ****mode***, va_list** *ap***);**
xbee_err xbee_validate(struct xbee ****xbee*);**
xbee_err xbee_shutdown(struct xbee ****xbee*);**

DESCRIPTION

The functions **xbee_setup**() and **xbee_vsetup**() start an instance of libxbee. *retXbee* is the returned pointer to the libxbee instance. *mode*specifies which mode should be started. Three modes are currently provided and supported:

'**xbee1**' the mode used for Series 1 XBee modules
'**xbee2**' the mode used for Series 2 XBee modules
'**xbee3**' the mode used for Series 3 XBee modules
'**xbee5**' the mode used for Series 5 XBee modules (868 MHz)
'**xbee6b**' the mode used for Series 6B XBee modules (Wi-Fi)
'**xbeeZB**' the mode used for ZigBee XBee modules
'**net**' the network client

'**debug**' the debugger

Each mode can require different initialization parameters, which are provided by the ... or *ap* arguments. The arguments for the modes listed above are detailed below in the 'Modes' section.

xbee_validate() allows you to confirm that your handle points to a valid libxbee instance.

xbee_shutdown() will terminate all remaining connections and free all data associated with the instance of libxbee.

Modes

'**xbee1**', '**xbee2**', '**xbee3**', '**xbee5**', '**xbee6b**' and '**xbeeZB**' - these modes require two parameters:

**char ***device e.g: "/dev/ttyUSB0"

int baudrate e.g: 57600

'**net**' - this mode required two parameters:

**char ***hostname e.g: "localhost", "127.0.0.1"

int port e.g: 27015

'**debug**' - this mode required one parameter:

**char ***target_mode e.g: "xbee1"

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

To setup libxbee with a Series 1 XBee, using /dev/ttyUSB0 at 57600 baud:

```
#include <xbee.h>

xbee_err ret;
struct xbee *xbee;

if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 57600)) != XBEE_ENONE) {
        printf("xbee_setup(): %d - %s\n", ret, xbee_errorToStr(ret));
        return ret;
}

/* use libxbee */

if (xbee_shutdown(xbee) != XBEE_ENONE) return;
```

NOTES

When running on OSX with a USB to UART adapter, you should choose **/dev/cu.usbserial-***, not **/dev/tty.usbserial-***.

XBEE_MODEGETLIST

NAME

xbee_modeGetList, xbee_modeGet

SYNOPSIS

#include <xbee.h>
xbee_err xbee_modeGetList(char ***retList);**
xbee_err xbee_modeGet(struct xbee ***xbee, const char ***mode**);**

DESCRIPTION

xbee_modeGetList() allows you to retrieve a list of modes that are build into libxbee. Once you have used the returned array, you should **free**() the pointer to avoid memory leaks. The last item in the array is set to **NULL** to mark the end.
xbee_modeGet() returnes the mode that the given libxbee instance is running. The pointer returned should *NOT* be **free**()'d.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

xbee_modeGetList()

```
#include <xbee.h>

char **modes;
int i;

if (xbee_modeGetList(&modes) != XBEE_ENONE) return;

for (i = 0; modes[i]; i++) {
        printf("mode %d: %s\n", i, modes[i]);
}

free(modes);
```

xbee_modeGet()

```
#include <xbee.h>

struct xbee *xbee;
char *mode;

/* initialize xbee, using xbee_setup() */
```

```
if (xbee_modeGet(xbee, &mode) != XBEE_ENONE) return;

printf("libxbee is running in '%s' mode\n", mode);
```

XBEE_DATAGET

NAME

xbee_dataGet, xbee_dataSet

SYNOPSIS

#include <xbee.h>
xbee_err xbee_dataSet(struct xbee **xbee*, void **newData*, void ***oldData*);
xbee_err xbee_dataGet(struct xbee **xbee*, void ***curData*);

DESCRIPTION

xbee_dataSet() allows you to associate any data you wish with a libxbee instance. This data will then be avaliable from anywhere that you have access to the instance, and works identically to the functions **xbee_conDataSet**(3)**and xbee_conDataGet**(3)**.**
xbee indicates which instance you would like to use, and *newData* indicates the data you wish to associate. The *oldData* parameter allows you to retrieve the data that was assigned before you assigned new data. If it is **NULL** then the data will not be returned.
The **xbee_dataGet**() function allows you to retrieve the current data, without assigning new data to the instance.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct my_struct my_info;

/* initialize xbee, using xbee_setup() */

/* keep scope in mind! if this function returns, the pointer will be invalid */
if (xbee_dataSet(xbee, &my_info, NULL) != XBEE_ENONE) return;
```

XBEE_CONADDRESS

NAME

xbee_conAddress

SYNOPSIS

#include <xbee.h>

```
struct xbee_conAddress {
        unsigned char addr16_enabled;
        unsigned char addr16[2];

        unsigned char addr64_enabled;
        unsigned char addr64[8];

        unsigned char endpoints_enabled;
        unsigned char endpoint_local;
        unsigned char endpoint_remote;

        unsigned char cluster_enabled;
        unsigned short cluster_id;

        unsigned char profile_enabled;
        unsigned short profile_id;
};
```

DESCRIPTION

The **xbee_conAddress** struct contains details on the network address for a remote node. It is recommended that you initialize it to zero by calling **memset**() before you fill it in and call **xbee_conNew**(3). The data is ordered with the MSB of the address, in the first byte of the array.
E.g: the 64-bit address 0x0013A200 0x40081826 would be initialized like so:

```
memset(&address, 0, sizeof(address));

address.addr64_enabled = 1;
address.addr64[0] = 0x00;
address.addr64[1] = 0x13;
address.addr64[2] = 0xA2;
address.addr64[3] = 0x00;
address.addr64[4] = 0x40;
address.addr64[5] = 0x08;
address.addr64[6] = 0x18;
```

address.addr64[7] = 0x26;
Endpoints are a feature of XBee Series 2 ZNet and ZigBee modules that are not present with XBee Series 1 modules.

XBEE_CONSETTINGS

NAME

xbee_conSettings

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conSettings(struct xbee_con **con*, struct xbee_conSettings **newSettings*, struct xbee_conSettings **oldSettings*);

```
struct xbee_conSettings {
        /* libxbee options: */
        unsigned char noBlock;
        unsigned char catchAll;
        unsigned char noWaitForAck;

        /* generic options: */
        unsigned char queueChanges; /* for AT connections */
        unsigned char disableAck;      /* specific options for XBee 1 / causes use of FrameID 0x00 for others */
        unsigned char broadcast;

        /* XBee 2 / ZNet options: */
        unsigned char multicast;

        /* XBee ZigBee options: */
        unsigned char disableRetries;
        unsigned char enableEncryption;
        unsigned char extendTimeout;

        /* XBee 5 options: */
        unsigned char noRoute;

        /* other */
        unsigned char broadcastRadius;
};
```

DESCRIPTION

xbee_conSettings() allows you to configure the connection's parameters.

One of *newSettings* and *oldSettings* can optionally be **NULL**, one must be non-NULL.
If they are non-NULL, then the struct pointed to by *oldSettings* will be filled with the effective settings before the call to **xbee_conSettings**(), and then *newSettings* will be used to update the connection's settings.

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct xbee_conSettings settings;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

if (xbee_conSettings(con, NULL, &settings) != XBEE_ENONE) return;
settings.disableAck = 1;
if (xbee_conSettings(con, &settings, NULL) != XBEE_ENONE) return;
```

XBEE_CONNEW

NAME

xbee_conNew, xbee_conValidate, xbee_conGetXBee, xbee_conEnd

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conNew(struct xbee ****xbee*, struct xbee_con *****retCon***, const char ****type***, struct xbee_conAddress ****address***);**
xbee_err xbee_conValidate(struct xbee_con ****con*);**
xbee_err xbee_conGetXBee(struct xbee_con ****con*, struct xbee *****xbee***);**
xbee_err xbee_conEnd(struct xbee_con ****con*);**

DESCRIPTION

xbee_conNew() will create new connection via the given libxbee instance *xbee*. The new connection will be returned via *retCon*, and the connection's type is provided by *type*. A list of avaliable connection types can be obtained by using **xbee_conGetTypes**(3). The *address*indicates which remote device you wish to communicate with - it is not always appropriate to use this (e.g: with a "Local AT" connection), in which case **NULL** may be safely provided.

xbee_conValidate() allows you to confirm that your handle points to a valid libxbee connection.
xbee_conGetXBee() allows you to retrieve the libxbee instance that was provided when the connection was created by calling **xbee_conNew**().
xbee_conEnd() will terminate a connection, and free any memory that is associated with it, including any un-fetched packets. If a callback is active when **xbee_conEnd**() is called, it will be permitted to complete, and then the connection will be terminated.

Avaliable Types

xbee1 "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Local AT" - Set/Get AT configuration from the local module.
"Remote AT" - Set/Get AT configuration from a remote module.
"16-bit Data" - 2-way communication with a remote module, using 16-bit addressing.
"64-bit Data" - 2-way communication with a remote module, using 64-bit addressing.
"16-bit I/O" - Rx Only. I/O samples received from a remote module, using 16-bit addressing.
"64-bit I/O" - Rx Only. I/O samples received from a remote module, using 64-bit addressing.
xbee2 "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Local AT" - Set/Get AT configuration from the local module.
"Remote AT" - Set/Get AT configuration from a remote module.
"Data" - 2-way communication with a remote module. All data will be recieved using this connection type if AO=0.
"Data (explicit)" - 2-way communication with a remote module, with configurable endpoints. All data will be recieved using this connection type if AO=1.
"I/O" - Rx Only. I/O samples received from a remote module. Only if AO=0.
"Sensor" - Rx Only. Used for receiving sensor samples. Only if AO=0.
"Identify" - Rx on the Coordinator Only. Identifies new nodes as they join the network. Only if AO=0.
xbee3 "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Local AT" - Set/Get AT configuration from the local module.
"Remote AT" - Set/Get AT configuration from a remote module.
"Data" - 2-way communication with a remote module. All data will be recieved using this connection type if AO=0.
"Data (explicit)" - 2-way communication with a remote module, with configurable endpoints. All data will be recieved using this connection type if AO=1.
"Identify" - Rx on the Coordinator Only. Identifies new nodes as they join the network. Only if AO=0.
xbee5 "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Local AT" - Set/Get AT configuration from the local module.

"Remote AT" - Set/Get AT configuration from a remote module.
"Data" - 2-way communication with a remote module. All data will be recieved using this connection type if AO=0.
"Data (explicit)" - 2-way communication with a remote module, with configurable endpoints. All data will be recieved using this connection type if AO=1.
"Identify" - Rx on the Coordinator Only. Identifies new nodes as they join the network. Only if AO=0.
xbee6b "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Frame Error" - Rx Only. Framing errors are received by this connection type, though they are currently of limited use because they aren't referenced to a specific frame...
"Local AT" - Set/Get AT configuration from the local module.
"Remote AT" - Set/Get AT configuration from a remote module.
"Data" - 2-way communication with a remote module.
"I/O" - Rx Only. I/O samples received from a remote module.
xbeeZB "Modem Status" - Rx Only. Status updates from the XBee module.
"Transmit Status" - Rx Only. Transmit status responses. These are handled for you, but you may wish to access them anyway.
"Local AT" - Set/Get AT configuration from the local module.
"Remote AT" - Set/Get AT configuration from a remote module.
"Data" - 2-way communication with a remote module. All data will be recieved using this connection type if AO=0.
"Data (explicit)" - 2-way communication with a remote module, with configurable endpoint, cluster and profile IDs. All data will be recieved using this connection type if AO=1.
"I/O" - Rx Only. I/O samples received from a remote module. Only if AO=0.
"Identify" - Rx on the Coordinator Only. Identifies new nodes as they join the network. Only if AO=0.
"Sensor" - Rx Only. Used for receiving sensor samples. Only if AO=0.
"OTA Update Status" - Rx on the Coordinator Only. Identifies new nodes as they join the network. Only if AO=0.
net the connection types for **net** mode are determined by the mode running on the server.
See **xbee_conGetTypes**(3).

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>). Postponed termination due to an active callback is considered a success.

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct xbee_conAddress address;
```

```
/* initialize xbee, using xbee_setup() */

memset(&address, 0, sizeof(address));
address.addr64_enabled = 1;
address.addr64[0] = 0x00;
address.addr64[1] = 0x13;
address.addr64[2] = 0xA2;
address.addr64[3] = 0x00;
address.addr64[4] = 0x40;
address.addr64[5] = 0x08;
address.addr64[6] = 0x18;
address.addr64[7] = 0x26;

if (xbee_conNew(xbee, &con, "64-bit Data", &address) != XBEE_ENONE) return;

/* make use of the connection... */

if (xbee_conEnd(con) != XBEE_ENONE) return;
```

XBEE_CONTX

NAME

xbee_conTx, xbee_convTx, xbee_connTx, xbee_conxTx, xbee_convxTx, xbee_connxTx

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conTx(struct xbee_con *_con_, unsigned char *_retVal_, const char *_format_, ...);
xbee_err xbee_convTx(struct xbee_con *_con_, unsigned char *_retVal_, const char *_format_, va_list _args_);
xbee_err xbee_connTx(struct xbee_con *_con_, unsigned char *_retVal_, const unsigned char *_buf_, int _len_);
xbee_err xbee_conxTx(struct xbee_con *_con_, unsigned char *_retVal_, unsigned char *_frameId_, const char *_format_, ...);
xbee_err xbee_convxTx(struct xbee_con *_con_, unsigned char *_retVal_, unsigned char *_frameId_, const char *_format_, va_list _args_);
xbee_err xbee_connxTx(struct xbee_con *_con_, unsigned char *_retVal_, unsigned char *_frameId_, const unsigned char *_buf_, int _len_);

DESCRIPTION

This collection of functions allow you to transmit data via an existing connection.
When calling **xbee_conTx**(), you may use a **printf**(3) style format string and parameters.
Similarly, **xbee_convTx**() makes use of **vprintf**(3). When using these two functions, the length of the message is determined from the return value of **vprintf**(3). This means that by using '**%c**' conversion

specifiers within the format string, you may achieve zero value - 0x00 - bytes before the end of your message. Do not use '**** ' as this will terminate the format string early

xbee_connTx() accepts a buffer and a length instead, allowing you to build the message yourself.

The 'x' variants namely **xbee_conxTx**(), **xbee_convxTx**() and **xbee_connxTx** can be used if you want to know what frame ID was used for the transmission.

If the connection type supports ACKs, and they are not disabled for the connection, and if *retVal* is non-NULL, it will be updated with the return value. E.g: 0x04 - No response for a Remote AT connection. You should consult the manual for the remote device for information on the values that this may return.

NOTE: not all connection types are able to transmit. For example the 'Modem Status', 'Transmit Status' and 'I/O' connections.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
xbee_err ret;
unsigned char retVal;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

if ((ret = xbee_conTx(con, &retVal, "Hello World! it is %d!", 2012)) != XBEE_ENONE) {
        if (ret == XBEE_ETX) {
                fprintf(stderr, "a transmission error occured... (0x%02X)\n", retVal);
        } else {
                fprintf(stderr, "an error occured... %s\n", xbee_errorToStr(ret));
        }
        return ret;
}
```

XBEE_CONRX

NAME

xbee_conRx

SYNOPSIS

#include <xbee.h>

xbee_err xbee_conRx(struct xbee_con *_con_, struct xbee_pkt **_retPkt_, int *_remainingPackets_);

xbee_err xbee_conRxWait(struct xbee_con *_con_, struct xbee_pkt **_retPkt_, int *_remainingPackets_);

DESCRIPTION

xbee_conRx() allows you to retrieve information from an active connection. *con* must be non-NULL, and must point at a valid connection. At least one of *retPkt* and *remainingPackets* must be non-NULL.
retPkt may optionally be non-NULL, in which case the next packet in the connection's buffer is returned to you. If you receive a packet from a connection, you must not forget to call **xbee_pktFree**(3) when you are finished with it. If *retPkt* is non-NULL, and a callback has been configured for the connection, then **XBEE_EINVAL** will be returned and neither *retPkt* or *remainingPackets* will have been updated - see**xbee_conCallbackSet**(3) for more information.
remainingPackets may also optionally be non-NULL. If it is non-NULL, then it will be populated with the number of packets that are currently in the connection's buffer.
When a callback is enabled, **xbee_conRx**() may be called with *retPkt* set to NULL, and *remainingPackets* set as non-NULL.
xbee_conRxWait() is identical to **xbee_conRx**(), except that it will busy-wait for upto 1 second, checking if there is a packet available.

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>).
XBEE_ENOTEXISTS will be returned if there is no packet available to retrieved.

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct xbee_pkt *pkt;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

/* receive some data for con */

if (xbee_conRx(con, &pkt, NULL) != XBEE_ENONE) return;

/* use pkt */

if (xbee_pktFree(pkt) != XBEE_ENONE) return;
```

XBEE_CONCALLBACKGET

NAME

xbee_conCallbackGet, xbee_conCallbackSet

SYNOPSIS

#include <xbee.h>
typedef void(*xbee_t_conCallback)(struct xbee *_xbee_, struct xbee_con *_con_, struct xbee_pkt **_pkt_, void **_data_);
xbee_err xbee_conCallbackSet(struct xbee_con *_con_, xbee_t_conCallback _newCallback_, xbee_t_conCallback *_oldCallback_);
xbee_err xbee_conCallbackGet(struct xbee_con *_con_, xbee_t_conCallback *_curCallback_);

DESCRIPTION

xbee_conCallbackSet() allows you to associate a callback function with a connection, whilst retrieving the current callback.
con must be a valid connection, returned from **xbee_conNew**(). *newCallback* must be either **NULL** (to disable callbacks), or non-NULL to enable callbacks, with the given function address. *oldCallback* can be **NULL** to indicate that you do not wish to retrieve the previously assigned callback.
xbee_conCallbackGet() allows you to retrieve the callback function that is currently assigned to the given connection.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

Note

While a connection has a callback function assigned, use of **xbee_conRx**() will be disabled, and it will return **XBEE_EINVAL**.

Callback Behavior

For each connection, one callback thread may be active. Within this thread, the connection's callback function will be executed once for each packet that is received, in the order that they were received. This means that if a single callback function is associated with multiple connections, static variables will be unsafe, and the developer should instead use the **void **_data_** that is provided. The *data* parameter is initialized to **NULL,** so you may safely initialize this from within the callback function, or alternatively from outside, by calling **xbee_conDataSet**().

Callback Parameters

The callback function's parameters are detailed below:

xbee

provides the libxbee instance from which the connection is derived.

con

provides the connection that the packet was received by (you may call a variant of **xbee_conTx**() with this from within the callback function).

pkt

provides access to the packet that was received.
NOTE: you must dereference this argument once. If you leave the argument alone, then libxbee will automatically call **xbee_pktFree**() on the packet after the callback has completed. If you instead re-assign **NULL** to it, libxbee will not call **xbee_pktFree**() on the packet. This is useful if you wish to postpone processing, in which case you must later call **xbee_pktFree**(). See the EXAMPLE section for more information.

data

provides the data item that you have assigned to the connection, either from within a callback, or by using **xbee_conDataSet**(). Again, you must dereference this argument once before use, and you may re-assign it from within the callback function, without calling**xbee_conDataSet**().

EXAMPLE

A simple callback

```
#include <xbee.h>

void myCallback(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data) {
        printf("received data for connection @ %p\n", con);
        xbee_conTx(con, "Hello!\n");
}

struct xbee *xbee;
struct xbee_con *con;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

if (xbee_conCallbackSet(con, myCallback, NULL) != XBEE_ENONE) return;
```

A callback that uses the connection's data

```
struct myStruct {
        int counter;
        struct xbee_pkt *pkt;
};
```

```
void myCallback(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data) {
	struct myStruct *ms;

	/* allocate storage for the counter & packet */
	if (!(*data)) {
		if ((ms = malloc(sizeof(*ms))) == NULL) return; /* error */
		/* keep hold of the storage */
		*data = ms;
	} else {
		ms = *data;
	}

	/* if we are already holding a packet, print a message and return (libxbee will free the packet) */
	if (ms->pkt) {
		printf("already holding a packet...\n");
		return;
	}

	/* otherwise increment the counter, and hold on to the packet */
	ms->counter++;
	ms->pkt = *pkt;

	printf("received %d packets for connection @ %p\n", ms->a, con);
	xbee_conTx(con, "Hello!\n");

	/* don't let libxbee free our packet */
	*pkt = NULL;
}

/* observe the data using xbee_conDataGet() */
```

XBEE_CONPURGE

NAME

xbee_conPurge

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conPurge(struct xbee_con *_con_);

DESCRIPTION

xbee_conPurge() allows you to discard ALL packets that are currently queued for a connection.

This function call has no effect if callbacks are enabled for the given connection.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

if (xbee_conPurge(con) != XBEE_ENONE) return;
```

XBEE_ATTACHEOFCALLBACK

NAME

xbee_attachEOFCallback

SYNOPSIS

#include <xbee.h>
xbee_err xbee_attachEOFCallback(struct xbee **xbee*, void (eofCallback*)(struct xbee **xbee*, void **rxInfo*));**

DESCRIPTION

xbee_attachEOFCallback() allows you to be notified if an EOF occurs on the device. An EOF can occur if a USB Serial Adaptor is unplugged, or if the network server closes the connection. It is useful to know that an EOF has occured so that you can take action (e.g: warn the user).

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

To configure an EOF callback:

```
#include <xbee.h>

void eek_eof(struct xbee *xbee, void *rxInfo) {
        printf("device failed - EOF\n");
        xbee_shutdown(xbee);
```

```
}

struct xbee *xbee;

/* initialize xbee, using xbee_setup() */

if (xbee_attachEOFCallback(xbee, eek_eof) != XBEE_ENONE) {
        /* alert of error */
}
```

XBEE_CONDATAGET

NAME

xbee_conDataGet, xbee_conDataSet

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conDataSet(struct xbee_con *_con_, void *_newData_, void **_oldData_);
xbee_err xbee_conDataGet(struct xbee_con *_con_, void **_curData_);

DESCRIPTION

xbee_conDataSet() allows you to associate any data you wish with a connection. This data will then be avaliable from within callback functions (see **xbee_conCallbackSet**(3) for more), or by calling **xbee_conDataGet**(3).
con indicates which connection you would like to use, and *newData* indicates the data you wish to associate.
The *oldData* parameter allows you to retrieve the data that was assigned before you assigned new data. If it is **NULL** then the data will not be returned.
The **xbee_conDataGet**() function allows you to retrieve the current data, without assigning new data to the connection.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct my_struct my_info;

/* initialize xbee, using xbee_setup() */
```

```
/* initialize con, using xbee_conNew() */

/* keep scope in mind! if this function returns, the pointer will be invalid */
if (xbee_conDataSet(con, &my_info, NULL) != XBEE_ENONE) return;
```

XBEE_CONINFOGET

NAME

xbee_conInfoGet, xbee_conInfo

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conInfoGet(struct xbee_con ****con*, struct xbee_conInfo ****info***);**
xbee_err xbee_conTypeGet(struct xbee_con ****con*, char *****type***);**

```
struct xbee_conInfo {
        int countRx;
        int countTx;

        time_t lastRxTime;
};
```

DESCRIPTION

xbee_conInfoGet() provides you with access to the statistical information stored with the connection. *info* should be pre-allocated storage to receive the *struct xbee_conInfo*.
xbee_conTypeGet() allows you to retrieve the type of the connection that was provided when calling **xbee_conNew**().

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct xbee_conInfo info;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */
```

```
if (xbee_conInfoGet(con, &info) != XBEE_ENONE) return;

printf("Tx count: %d\n", info.countTx);
printf("Rx count: %d\n", info.countRx);
printf("Last Rx was at: %s\n", ctime(&info.lastRxTime));
```

XBEE_CONGETTYPES

NAME

xbee_conGetTypes

SYNOPSIS

#include <xbee.h>
xbee_err xbee_conGetTypes(struct xbee *_xbee_, char ***_retList_);

DESCRIPTION

xbee_conGetTypes() allows you to retrieve a list of connection types that are provided by the given libxbee instance's mode. Once you have used the returned array, you should **free**() the pointer to avoid memory leaks. The last item in the array is set to **NULL** to mark the end.
For a list of connection types generally avaliable, please see **xbee_conNew**(3).

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
char **types;
int i;

/* initialize xbee, using xbee_setup() */

if (xbee_conGetTypes(xbee, &types) != XBEE_ENONE) return;

for (i = 0; types[i]; i++) {
        printf("type %d: %s\n", i, types[i]);
}

free(types);
```

XBEE_PKT

NAME

xbee_pkt

SYNOPSIS

#include <xbee.h>

```
struct xbee_pkt {
        struct xbee *xbee;
        struct xbee_con *con;
        const char *conType;

        unsigned char status;
        unsigned char options;
        unsigned char rssi; /* print as "RSSI: -%d - only valid for XBee 1 */
        unsigned char apiIdentifier;
        unsigned char frameId;

        struct timespec timestamp;

        struct xbee_conAddress address;

        unsigned char atCommand[2];

        void *dataItems; /* do NOT access this item */

        int dataLen;
        unsigned char data[1];
};
```

DESCRIPTION

The *xbee_pkt* structure provides access to the information contained within a packet.

xbee

provides a handle to the related libxbee instance

con

provides a handle to the related libxbee connection

status

provides the returned status information. This is not populated for all connection types, but is populated for **Local AT**, **Remote AT** and possibly others (refer to the XBee datasheet for more information).

options

like *status*, this field isn't populated for every connection type, but is populated for **Data** and **I/O** (refer to the XBee datasheet for more information).

rssi

this field contains the 'Received Signal Strength Indication' value.

If populated, this can be printed like so:

printf("RSSI: -%d dBm\n", pkt->rssi);

apiIdentifier

contains the API Identifier for the XBee message type that was handled

frameId

contains the Frame ID provided by the received message

atCommand

this is only populated for AT connection types, and contains the 2 character AT command.

dataItems

you should not dereference this pointer, it is for internal use *only*. To access data, use **xbee_pktDataGet**(3) and its variations.

dataLen

this indicates the length of the *data* field.

data

this contains the body of the packet. This can contain **zero** bytes, and libxbee will nul terminate it, so that data may be treated as a string (using **printf**(3) for example).

XBEE_PKTVALIDATE

NAME

xbee_pktValidate, xbee_pktFree

SYNOPSIS

#include <xbee.h>

xbee_err xbee_pktValidate(struct xbee_pkt **pkt*);

xbee_err xbee_pktFree(struct xbee_pkt **pkt*);

DESCRIPTION

These functions allow you to confirm your handle points to a valid libxbee packet, and free the memory associated with a packet.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
```

```
struct xbee_con *con;
struct xbee_pkt *pkt;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

/* receive some data */

if (xbee_conRx(con, &pkt, NULL) != XBEE_ENONE) return;

if (xbee_pktValidate(pkt) != XBEE_ENONE) return;

/* use the packet's contents */

if (xbee_pktFree(pkt) != XBEE_ENONE) return;
```

XBEE_PKTDATAGET

NAME

xbee_pktDataGet, xbee_pktAnalogGet, xbee_pktDigitalGet

SYNOPSIS

#include <xbee.h>
xbee_err xbee_pktDataGet(struct xbee_pkt * *pkt*, **const char *** *key*, **int** *id*, **int** *index*, **void ****retData*);
xbee_err xbee_pktAnalogGet(struct xbee_pkt * *pkt*, **int** *channel*, **int** *index*, **int *** *retVal*);
xbee_err xbee_pktDigitalGet(struct xbee_pkt * *pkt*, **int** *channel*, **int** *index*, **int *** *retVal*);

DESCRIPTION

These functions provide access to data that has been parsed out of the packet's body by libxbee. In general, the raw data is still accessible though the packet, but this creates a more friendly interface.
xbee_pktDataGet() will search the packet's *dataItems* (see **xbee_pkt**(3) for more) for a named *key*.
The *id* and *index* are used to give a 2D-esque layout to the *key*. However if *id* is given as -1, then it is ignored, and the first 'column' is used. This allows for similar data - e.g: analog samples - to be stored using a single key, with differing channel numbers (*id*'s).
xbee_pktAnalogGet(w,x,y,&z) is nearly synonymous with **xbee_pktDataGet**(w,analog,x,y,&z), though due to internal magic, it is strongly recommended that you use **xbee_pktAnalogGet**().
Similarly, **xbee_pktDigitalGet**(w,x,y,&z) is nearly synonymous with **xbee_pktDataGet**(w,digital,x,y,&z).

Keys

Below is a list of keys that libxbee uses, 'analog' and 'digital' are not included, because you should use the relative functions instead.

The list is keyed - <xbee mode>:<connection type>
xbee2:Identify
"Address (16-bit)" - the raw 16-bit address
"Address (64-bit)" - the raw 64-bit address
"Address" - a *struct xbee_conAddress* containing the full address information. This is not stored within the packet's data field
"NI" - the node identifier

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;
struct xbee_pkt *pkt;
int value;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() to an I/O connection */

/* receive a packet with analog I/O data, using xbee_conRx() */

if (xbee_pktAnalogGet(pkt, 3, 0, &value) != XBEE_ENONE) return;

/* 'value' now contains the ADC value from the A3 pin */
```

XBEE_CONSLEEPGET

NAME

xbee_conSleepGet, xbee_conSleepSet, xbee_conSleepStates

SYNOPSIS

```
#include <xbee.h>
enum xbee_conSleepStates {
        CON_AWAKE,
        CON_SNOOZE,
        CON_SLEEP
};
```

xbee_err xbee_conSleepSet(struct xbee_con *_con_, enum xbee_conSleepStates _state_);
xbee_err xbee_conSleepGet(struct xbee_con *_con_, enum xbee_conSleepStates *_state_);

DESCRIPTION

xbee_conSleepSet() allows you to put a connection to sleep. If a connection is sleeping, another connection may be created with the same address.
If a connection is set to **CON_SNOOZE** then inbound packets, and calling **xbee_conTx**(3) will wake it up *ONLY* if there are no other awake connections with the same address.
If a connection is set to **CON_SLEEP** then inbound packets will not wake it up, and it must be explicitly woken before **xbee_conTx**(3) will succeed.
Waking a connection will *ONLY* succeed if there are no other connections that are currently awake that share the same address.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
struct xbee_con *con;

/* initialize xbee, using xbee_setup() */

/* initialize con, using xbee_conNew() */

if (xbee_conSleepSet(con, &my_info, NULL) != XBEE_ENONE) return;
```

XBEE_NETSTART

NAME

xbee_netStart, xbee_netvStart, xbee_netStop

SYNOPSIS

#include <xbee.h>
xbee_err xbee_netStart(struct xbee *_xbee_, int _port_, int(*_clientFilter_)(struct xbee *xbee, const char *remoteHost));
xbee_err xbee_netvStart(struct xbee *_xbee_, int _fd_, int(*_clientFilter_)(struct xbee *xbee, const char *remoteHost));
xbee_err xbee_netStop(struct xbee *_xbee_);

DESCRIPTION

These functions allow you to start and stop the network interface for libxbee.

If you use **xbee_netStart**() libxbee will listen on all avaliable interfaces, you can only specify which port to listen on.
If you use **xbee_netvStart**() you must provide a file descriptor for a socket that has been bound to an interface and port - using **socket**(2) and**bind**(2) - but is not yet listening.
For both of these functions, you may specify a *clientFilter* that may decide if the client will be granted access, or denied. If the function returns 0, then the connection will be permitted, otherwise it will be terminated. See the example for more details.
If you wish to terminate the network interface before calling **xbee_shutdown**(3), you may use **xbee_netStop**(). If you do not call**xbee_netStop**() before **xbee_shutdown**(3), then libxbee will call it for you.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

int myClientFilter(struct xbee *xbee, const char *remoteHost) {
        printf("Connection request from [%s]\n", remoteHost);
        return 0; /* return 0 - accept, anything else means deny    */
}

struct xbee *xbee;

/* initialize xbee, using xbee_setup() */

xbee_netStart(xbee, 27015, myClientFilter);

/* use the network interface... */

xbee_netStop(xbee);
```

XBEE_LOG

NAME

xbee_log

SYNOPSIS

#include <xbee.h>
xbee_err xbee_log(struct xbee ****xbee*, int** *minLevel***, const char ****format***, ...);**

DESCRIPTION

xbee_log() provides a means for you to write messages to libxbee's log.

xbee

is required, and will indicate which log to use

minLevel

is a value that determines whether the message will be output or not. The lower this value, the more likely. libxbee defaults to a log level of 0, but this may be changed by specifying **XBEE_LOG_LEVEL** at build time, by specifying **XBEE_LOG_LEVEL** in the environment at run-time, or by calling **xbee_logLevelSet**(3) at run-time. These options are applied in this order.

format, ...

these options are exactly the same as those used by **printf**(3). You do not need to add a newline ('\n') character to the end, libxbee will do this for you.

Return Value

On success this function will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;

/* initialize xbee, using xbee_setup() */

xbee_log(xbee, -1, "Hello World!");
```

Output

DEV: -1#[main.c:14] main() 0x82dd128: Hello World!

Where:

DEV:

indicates that this was a message logged from outside libxbee (within your program)

-1

indicates the log level for the message

[main.c:14]

indicates the filename and line number

main()

indicates the function that the call to **xbee_log**() was made from

0x82dd128

indicates the address of the libxbee instance

Hello World!

is the message

XBEE_LOGLEVELGET

NAME

xbee_logLevelGet, xbee_logLevelSet, xbee_logTargetGet, xbee_logTargetSet xbee_logRxGet, xbee_logRxSet, xbee_logTxGet, xbee_logTxSet, xbee_logColorGet, xbee_logColorSet

SYNOPSIS

#include <xbee.h>
xbee_err xbee_logLevelGet(struct xbee *_xbee_, int *_level_);
xbee_err xbee_logLevelSet(struct xbee *_xbee_, int _level_);
xbee_err xbee_logTargetGet(struct xbee *_xbee_, FILE **_f_);
xbee_err xbee_logTargetSet(struct xbee *_xbee_, FILE *_f_);
xbee_err xbee_logRxGet(struct xbee *_xbee_, int *_enable_);
xbee_err xbee_logRxSet(struct xbee *_xbee_, int _enabled_);
xbee_err xbee_logTxGet(struct xbee *_xbee_, int *_enable_);
xbee_err xbee_logTxSet(struct xbee *_xbee_, int _enabled_);
xbee_err xbee_logColorGet(struct xbee *_xbee_, int *_enable_);
xbee_err xbee_logColorSet(struct xbee *_xbee_, int _enabled_);

DESCRIPTION

These functions allow you to configure and retrieve the logging options for an instance of libxbee.
The *level* may be any number, and indicates the verbosity of log messages. A higher number will generate a more verbose output. See**xbee_log**(3) for more details on how this affects log output.
You may redirect the log output using **xbee_logTargetSet**(). *f* must be an open file descriptor that will allow writing. By default, libxbee will log to stderr.
You may enable and disable various logging components using **xbee_logRxSet**()
and **xbee_logTxSet**(). *enable* is handled as a boolean value, and will enable or disable logging of either Rx data or Tx payload data respectively.
You may also enable or disable colorized logging using **xbee_logColorSet**(). *enable* is handled as a boolean value, and will enable or disable the output of ANSI color escape sequences. Colorized output only works for TTY devices, not files. if you use **xbee_logTargetSet**() to log to a file, then the output will not contain ANSI color escape sequences.

Return Value

On success these functions will return XBEE_ENONE, otherwise an error number from *enum xbee_errors* (as specified in <*xbee.h*>)

NOTES

Some of these functions may return **XBEE_ENOTIMPLEMENTED** if libxbee has bee compiled with them disabled.
These settings may be configured at runtime using the following environment variables:
XBEE_LOG_LEVEL

XBEE_LOG_RX
XBEE_LOG_TX
XBEE_LOG_COLOR

EXAMPLE

```
#include <xbee.h>

struct xbee *xbee;
FILE *log;

/* initialize xbee, using xbee_setup() */

if ((log = fopen("libxbee.log", "w")) == NULL) return;

if (xbee_logTargetSet(xbee, log) != XBEE_ENONE) return;

if (xbee_logLevelSet(xbee, 100) != XBEE_ENONE) return;
```

XBEE_ERRORTOSTR

NAME

xbee_errorToStr, xbee_errors, xbee_err

SYNOPSIS

```
#include <xbee.h>
const char *xbee_errorToStr(xbee_err error);
enum xbee_errors {
        XBEE_ENONE                      =        0,
        XBEE_EUNKNOWN                   =       -1,

        XBEE_ENOMEM                     =       -2,

        XBEE_ESELECT                    =       -3,
        XBEE_ESELECTINTERRUPTED         =       -4,

        XBEE_EEOF                       =       -5,
        XBEE_EIO                        =       -6,

        XBEE_ESEMAPHORE                 =       -7,
        XBEE_EMUTEX                     =       -8,
        XBEE_ETHREAD                    =       -9,
```

```
        XBEE_ELINKEDLIST        =       -10,

        XBEE_ESETUP             =       -11,
        XBEE_EMISSINGPARAM      =       -12,
        XBEE_EINVAL             =       -13,
        XBEE_ERANGE             =       -14,
        XBEE_ELENGTH            =       -15,

        XBEE_EFAILED            =       -18,
        XBEE_ETIMEOUT           =       -17,
        XBEE_EWOULDBLOCK        =       -16,
        XBEE_EINUSE             =       -19,
        XBEE_EEXISTS            =       -20,
        XBEE_ENOTEXISTS         =       -21,
        XBEE_ENOFREEFRAMEID     =       -22,

        XBEE_ESTALE             =       -23,
        XBEE_ENOTIMPLEMENTED    =       -24,

        XBEE_ETX                =       -25,

        XBEE_EREMOTE            =       -26,

        XBEE_ESLEEPING          =       -27,
        XBEE_ECATCHALL          =       -28,

        XBEE_ESHUTDOWN          =       -29,
};
```

typedef enum xbee_errors xbee_err;

DESCRIPTION

xbee_errorToStr() provides simple error number to textual error message resolution. The returned string should not be **free**()'d and is not modifiable.

This function behaves similarly to **strerror**(3) but the returned string will not be modified by subsequent calls.

Return Value

On success this function will return a pointer to a nul terminated string, otherwise a generic error message will be returned.

LIBXBEE_REVISION

NAME

libxbee_revision, libxbee_commit, libxbee_committer, libxbee_buildtime

SYNOPSIS

#include <xbee.h>
extern const char libxbee_revision[];
extern const char libxbee_commit[];
extern const char libxbee_committer[];
extern const char libxbee_buildtime[];

DESCRIPTION

These are static strings that are compiled into the library. These are intended to help with debugging (when you contact me).

libxbee_revision

contains a version string
e.g: "v3.0.4"

libxbee_commit

contains the Git commit ID that the library was built from.
e.g: "9e8da59e41a6e901d2568861ea6323119927c9cc"
This string is used to ensure compatibility between the network server and clients. This check can be disabled by specifying**XBEE_NO_NET_STRICT_VERSIONS** at build time.

libxbee_committer

contains the name and email of the committer
e.g: "Attie Grande <attie@attie.co.uk>"

libxbee_buildtime

contains the buildtime, as output by `date`
e.g: "Sun Mar 4 17:43:26 GMT 2012"

EXAMPLE

```
#include <xbee.h>

printf("Libxbee revision: %s\n",      libxbee_revision);
printf("Libxbee commit: %s\n",       libxbee_commit);
printf("Libxbee committer: %s\n", libxbee_committer);
printf("Libxbee build time: %s\n",  libxbee_buildtime);
```