NAME

mio_open, **mio_close**, **mio_read**, **mio_write**, **mio_nfds**, **mio_pollfd**, **mio_revents**, **mio_eof** — sndio interface to MIDI streams

SYNOPSIS

#include <sndio.h>

struct mio_hdl ∗
mio_open(*const char ∗name*, *unsigned int mode*, *int nbio_flag*);

void
mio_close(*struct mio_hdl ∗hdl*);

size_t
mio_read(*struct mio_hdl ∗hdl*, *void ∗addr*, *size_t nbytes*);

size_t
mio_write(*struct mio_hdl ∗hdl*, *const void ∗addr*, *size_t nbytes*);

int
mio_nfds(*struct mio_hdl ∗hdl*);

int
mio_pollfd(*struct mio_hdl ∗hdl*, *struct pollfd ∗pfd*, *int events*);

int
mio_revents(*struct mio_hdl ∗hdl*, *struct pollfd ∗pfd*);

int
mio_eof(*struct mio_hdl ∗hdl*);

DESCRIPTION

The **sndio** library allows user processes to access `midi`(4) hardware and `sndiod`(8) MIDI thru boxes and control ports in a uniform way.

Opening and closing an MIDI stream

First the application must call the **mio_open**() function to obtain a handle representing the newly created stream; later it will be passed as the *hdl* argument of most other functions. The *name* parameter gives the device string discussed in `sndio`(7). If the program is using a single device and is providing no device chooser, it should be set to MIO_PORTANY to allow the user to select it using the `MIDIDEVICE` environment variable.

The *mode* parameter gives the direction of the stream. The following are supported:

MIO_OUT	The stream is output-only; data written to the stream will be sent to the hardware or other programs.
MIO_IN	The stream is input-only; received data from the hardware or other programs must be read from the stream.
MIO_IN | MIO_OUT	The stream sends and receives data. This mode should be used rather than calling **mio_open**() twice.

If the *nbio_flag* argument is true (i.e. non-zero), then the **mio_read**() and **mio_write**() functions (see below) will be non-blocking.

The **mio_close**() function closes the stream and frees all allocated resources associated with the **libsndio** handle.

Sending and receiving data

When input mode is selected, the **mio_read**() function must be called to retrieve received data; it must be called often enough to ensure that internal buffers will not overrun. It will store at most *nbytes* bytes at the *addr* location. Unless the *nbio_flag* flag is set, it will block until data becomes available and will return zero only on error.

When output mode is selected, the **`mio_write`**() function can be called to provide data to transmit. Unless the *`nbio_flag`* is set, **`mio_write`**() will block until the requested amount of data is written.

Non-blocking mode operation

If the *`nbio_flag`* is set on **`mio_open`**(), then the **`mio_read`**() and **`mio_write`**() functions will never block; if no data is available, they will return zero immediately.

To avoid busy loops when non-blocking mode is used, the `poll`(2) system call can be used to check if data can be read from or written to the stream. The **`mio_pollfd`**() function prepares the array *`pfd`* of *pollfd* structures for use with `poll`(2). The optimal size of the *`pfd`* array, which the caller must pre-allocate, is provided by the **`mio_nfds`**() function.

`poll`(2) will sleep until any of the *`events`* requested with **`mio_pollfd`**() have occurred. Events are represented as a bit-mask of *POLLIN* and *POLLOUT* constants. The events which woke up `poll`(2) can be obtained with the **`mio_revents`**() function. If *POLLIN* is set, **`mio_read`**() can be called without blocking. If *POLLOUT* is set, **`mio_write`**() can be called without blocking. POLLHUP may be set if an error occurs, even if it is not requested with **`mio_pollfd`**().

Error handling

Errors related to the MIDI subsystem (like hardware errors or dropped connections) and programming errors (such as a call to **`mio_read`**() on a play-only stream) are considered fatal. Once an error occurs, all functions which take a *mio_hdl* argument, except **`mio_close`**() and **`mio_eof`**(), stop working (i.e. always return 0).

RETURN VALUES

The **`mio_open`**() function returns the newly created handle on success or NULL on failure.

The **`mio_pollfd`**() function returns the number of *pollfd* structures filled. The **`mio_nfds`**() function returns the number of *pollfd* structures the caller must preallocate in order to be sure that **`mio_pollfd`**() will never overrun.

The **`mio_revents`**() function returns the bit-mask set by `poll`(2) in the *pfd* array of *pollfd* structures.

The **`mio_read`**() and **`mio_write`**() functions return the number of bytes transferred.

The **`mio_eof`**() function returns 0 if there's no pending error, and a non-zero value if there's an error.

ENVIRONMENT

`SNDIO_DEBUG`	The debug level: may be a value between 0 and 2.

SEE ALSO

`poll`(2), `midi`(4), `sndio`(7), `sndiod`(8)