NAME

mibII, **mibif_notify_f**, **mib_netsock**, **mib_if_set_dyn**, **mib_refresh_iflist**, **mib_find_if**, **mib_find_if_sys**, **mib_find_if_name**, **mib_first_if**, **mib_next_if**, **mib_register_newif**, **mib_unregister_newif**, **mib_fetch_ifmib**, **mib_if_admin**, **mib_find_ifa**, **mib_first_ififa**, **mib_next_ififa**, **mib_ifstack_create**, **mib_ifstack_delete**, **mib_find_rcvaddr**, **mib_rcvaddr_create**, **mib_rcvaddr_delete**, **mibif_notify**, **mibif_unnotify** - mib-2 module for bsnmpd.

LIBRARY

(begemotSnmpdModulePath."mibII" = /usr/lib/snmp_mibII.so)

SYNOPSIS

#include <net/if.h>
#include <net/if_mib.h>
#include <bsnmp/snmpmod.h>
#include <bsnmp/snmp_mibII.h>

typedef void
(*mibif_notify_f)(*struct mibif *ifp*, *enum mibif_notify event*, *void *uarg*);

extern int mib_netsock;

void
mib_if_set_dyn(*const char *ifname*);

void
mib_refresh_iflist(*void*);

*struct mibif **
mib_find_if(*u_int ifindex*);

*struct mibif **
mib_find_if_sys(*u_int sysindex*);

*struct mibif **
mib_find_if_name(*const char *ifname*);

*struct mibif **
mib_first_if(*void*);

*struct mibif **

mib_next_if(*const struct mibif *ifp*);

int
mib_register_newif(*int (*func)(struct mibif *), const struct lmodule *mod*);

void
mib_unregister_newif(*const struct lmodule *mod*);

int
mib_fetch_ifmib(*struct mibif *ifp*);

int
mib_if_admin(*struct mibif *ifp, int up*);

*struct mibifa **
mib_find_ifa(*struct in_addr ipa*);

*struct mibifa **
mib_first_ififa(*const struct mibif *ifp*);

*struct mibifa **
mib_next_ififa(*struct mibifa *ifa*);

int
mib_ifstack_create(*const struct mibif *lower, const struct mibif *upper*);

void
mib_ifstack_delete(*const struct mibif *lower, const struct mibif *upper*);

*struct mibrcvaddr **
mib_find_rcvaddr(*u_int ifindex, const u_char *addr, size_t addrlen*);

*struct mibrcvaddr **
mib_rcvaddr_create(*struct mibif *ifp, const u_char *addr, size_t addrlen*);

void
mib_rcvaddr_delete(*struct mibrcvaddr *addr*);

*void **
mibif_notify(*struct mibif *ifp, const struct lmodule *mod, mibif_notify_f func, void *uarg*);

void
mibif_unnotify(*void *reg*);

DESCRIPTION

The **snmp_mibII** module implements parts of the internet standard MIB-2. Most of the relevant MIBs are implemented. Some of the tables are restricted to be read-only instead of read-write. The exact current implementation can be found in */usr/share/snmp/defs/mibII_tree.def*. The module also exports a number of functions and global variables for use by other modules, that need to handle network interfaces. This man page describes these functions.

DIRECT NETWORK ACCESS

The **mibII** module opens a socket that is used to execute all network related ioctl(2) functions. This socket is globally available under the name *mib_netsock*.

NETWORK INTERFACES

The **mibII** module handles a list of all currently existing network interfaces. It allows other modules to handle their own interface lists with special information by providing a mechanism to register to events that change the interface list (see below). The basic data structure is the interface structure:

```
struct mibif {
        TAILQ_ENTRY(mibif) link;
        u_int              flags;
        u_int              index;     /* logical ifindex */
        u_int              sysindex;
        char               name[IFNAMSIZ];
        char               descr[256];
        struct ifmibdata mib;
        uint64_t   mibtick;
        void               *specmib;
        size_t             specmiblen;
        u_char             *physaddr;
        u_int              physaddrlen;
        int                has_connector;
        int                trap_enable;
        uint64_t   counter_disc;
        mibif_notify_f     xnotify;
        void               *xnotify_data;
        const struct lmodule *xnotify_mod;
        struct asn_oid     spec_oid;
};
```

The **mibII** module tries to implement the semantic if *ifIndex* as described in RFC-2863. This RFC states, that an interface indexes may not be reused. That means, for example, if *tun* is a synthetic interface type and the system creates the interface *tun0*, destroys this interfaces and again creates a *tun 0*, then these interfaces must have different interface indexes, because in fact they are different interfaces. If, on the other hand, there is a hardware interface *xl0* and this interface disappears, because its driver is unloaded and appears again, because the driver is loaded again, the interface index must stay the same. **mibII** implements this by differentiating between real and synthetic (dynamic) interfaces. An interface type can be declared dynamic by calling the function **mib_if_set_dyn**() with the name if the interface type (for example "tun)." For real interfaces, the module keeps the mapping between the interface name and its *ifIndex* in a special list, if the interface is unloaded. For dynamic interfaces a new *ifIndex* is generated each time the interface comes into existence. This means, that the interface index as seen by SNMP is not the same index as used by the system. The SNMP *ifIndex* is held in field *index*, the system's interface index is *sysindex*.

A call to **mib_refresh_iflist** causes the entire interface list to be re-created.

The interface list can be traversed with the functions **mib_first_if**() and **mib_next_if**(). Be sure not to change the interface list while traversing the list with these two calls.

There are three functions to find an interface by name or index. **mib_find_if**() finds an interface by searching for an SNMP *ifIndex*, **mib_find_if_sys**() finds an interface by searching for a system interface index and **mib_find_if_name**() finds an interface by looking for an interface name. Each of the function returns NULL if the interface cannot be found.

The function **mib_fetch_ifmib**() causes the interface MIB to be refreshed from the kernel.

The function **mib_if_admin**() can be used to change the interface administrative state to up (argument is 1) or down (argument is 0).

INTERFACE EVENTS

A module can register itself to receive a notification when a new entry is created in the interface list. This is done by calling **mib_register_newif**(). A module can register only one function, a second call to **mib_register_newif**() causes the registration to be overwritten. The registration can be removed with a call to **mib_unregister_newif**(). It is unregistered automatically, when the registering module is unloaded.

A module can also register to events on a specific interface. This is done by calling **mibif_notify**(). This causes the given callback *func* to be called with the interface pointer, a notification code and the user argument *uarg* when any of the following events occur:

MIBIF_NOTIFY_DESTROY
The interface is destroyed.

This mechanism can be used to implement interface type specific MIB parts in other modules. The registration can be removed with **mib_unnotify**() which the return value from *mib_notify*. Any notification registration is removed automatically when the interface is destroyed or the registering module is unloaded. *Note that only one module can register to any given interface.*

INTERFACE ADDRESSES

The **mibII** module handles a table of interface IP-addresses. These addresses are held in a

```
struct mibifa {
        TAILQ_ENTRY(mibifa) link;
        struct in_addr      inaddr;
        struct in_addr      inmask;
        struct in_addr      inbcast;
        struct asn_oid      index;
        u_int               ifindex;
        u_int               flags;
};
```

The (ordered) list of IP-addresses on a given interface can be traversed by calling **mib_first_ififa**() and **mib_next_ififa**(). The list should not be considered read-only.

INTERFACE RECEIVE ADDRESSES

The internet MIB-2 contains a table of interface receive addresses. These addresses are handled in:

```
struct mibrcvaddr {
        TAILQ_ENTRY(mibrcvaddr) link;
        struct asn_oid      index;
        u_int               ifindex;
        u_char              addr[ASN_MAXOIDLEN];
        size_t              addrlen;
        u_int               flags;
};
enum {
        MIBRCVADDR_VOLATILE         = 0x00000001,
        MIBRCVADDR_BCAST    = 0x00000002,
        MIBRCVADDR_HW               = 0x00000004,
};
```

Note, that the assignment of MIBRCVADDR_BCAST is based on a list of known interface types. The flags should be handled by modules implementing interface type specific MIBs.

A receive address can be created with **mib_rcvaddr_create**() and deleted with **mib_rcvaddr_delete**(). This needs to be done only for addresses that are not automatically handled by the system.

A receive address can be found with **mib_find_rcvaddr**().

INTERFACE STACK TABLE

The **mibII** module maintains also the interface stack table. Because for complex stacks, there is no system supported generic way of getting this information, interface type specific modules need to help setting up stack entries. The **mibII** module handles only the top and bottom entries.

A table entry is created with **mib_ifstack_create**() and deleted with **mib_ifstack_delete**(). Both functions need the pointers to the interfaces. Entries are automatically deleted if any of the interfaces of the entry is destroyed. The functions handle both the stack table and the reverse stack table.

FILES

/usr/share/snmp/defs/mibII_tree.def	The description of the MIB tree implemented by **mibII**.
/usr/local/share/snmp/mibs	
/usr/share/snmp/mibs/	The various internet MIBs.

SEE ALSO

gensnmptree(1), snmpmod(3)

STANDARDS

This implementation conforms to the applicable IETF RFCs.

AUTHORS

Hartmut Brandt <harti@FreeBSD.org>