NAME

snmp_netgraph, **snmp_node**, **snmp_nodename**, **ng_cookie_f**, **ng_hook_f**, **ng_register_cookie**, **ng_unregister_cookie**, **ng_register_hook**, **ng_unregister_hook**, **ng_unregister_module**, **ng_output**, **ng_output_node**, **ng_output_id**, **ng_dialog**, **ng_dialog_node**, **ng_dialog_id**, **ng_send_data**, **ng_mkpeer_id**, **ng_connect_node**, **ng_connect_id**, **ng_connect2_id**, **ng_connect2_tee_id**, **ng_rmhook**, **ng_rmhook_id**, **ng_rmhook_tee_id**, **ng_shutdown_id**, **ng_next_node_id**, **ng_node_id**, **ng_node_id_node**, **ng_node_name**, **ng_node_type**, **ng_peer_hook_id** **-** netgraph module for snmpd

LIBRARY

(begemotSnmpdModulePath."netgraph" = /usr/lib/snmp_netgraph.so)

SYNOPSIS

#include <bsnmp/snmpmod.h>
#include <bsnmp/snmp_netgraph.h>

extern ng_ID_t snmp_node;
*extern u_char *snmp_nodename*;

typedef void
ng_cookie_f(*const struct ng_mesg *mesg*, *const char *path*, *ng_ID_t id*, *void *uarg*);

typedef void
ng_hook_f(*const char *hook*, *const u_char *mesg*, *size_t len*, *void *uarg*);

*void **
ng_register_cookie(*const struct lmodule *mod*, *uint32_t cookie*, *ng_ID_t id*, *ng_cookie_f *func*, *void *uarg*);

void
ng_unregister_cookie(*void *reg*);

*void **
ng_register_hook(*const struct lmodule *mod*, *const char *hook*, *ng_hook_f *func*, *void *uarg*);

void
ng_unregister_hook(*void *reg*);

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

int
ng_output(*const char *path*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

int
ng_output_node(*const char *node*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

int
ng_output_id(*ng_ID_t node*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

*struct ng_mesg **
ng_dialog(*const char *path*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

*struct ng_mesg **
ng_dialog_node(*const char *node*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

*struct ng_mesg **
ng_dialog_id(*ng_ID_t id*, *u_int cookie*, *u_int opcode*, *const void *arg*, *size_t arglen*);

int
ng_send_data(*const char *hook*, *const void *sndbuf*, *size_t sndlen*);

ng_ID_t
ng_mkpeer_id(*ng_ID_t id*, *const char *name*, *const char *type*, *const char *hook*, *const char *peerhook*);

int
ng_connect_node(*const char *node*, *const char *ourhook*, *const char *peerhook*);

int
ng_connect_id(*ng_ID_t id*, *const char *ourhook*, *const char *peerhook*);

int
ng_connect2_id(*ng_ID_t id*, *ng_ID_t peer*, *const char *ourhook*, *const char *peerhook*);

int
ng_connect2_tee_id(*ng_ID_t id*, *ng_ID_t peer*, *const char *ourhook*, *const char *peerhook*);

int
ng_rmhook(*const char *ourhook*);

int

ng_rmhook_id(*ng_ID_t id*, *const char *hook*);

int
ng_rmhook_tee_id(*ng_ID_t id*, *const char *hook*);

int
ng_shutdown_id(*ng_ID_t id*);

ng_ID_t
ng_next_node_id(*ng_ID_t node*, *const char *type*, *const char *hook*);

ng_ID_t
ng_node_id(*const char *path*);

ng_ID_t
ng_node_id_node(*const char *node*);

ng_ID_t
ng_node_name(*ng_ID_t id*, *char *name*);

ng_ID_t
ng_node_type(*ng_ID_t id*, *char *type*);

int
ng_peer_hook_id(*ng_ID_t id*, *const char *hook*, *char *peerhook*);

DESCRIPTION

The **snmp_netgraph** module implements a number of tables and scalars that enable remote access to the netgraph subsystem. It also exports a number of global variables and functions, that allow other modules to easily use the netgraph system.

If upon start up of the module the variable *begemotNgControlNodeName* is not empty the module opens a netgraph socket and names that socket node. If the variable is empty, the socket is created, as soon as the variable is written with a non-empty name. The socket can be closed by writing an empty string to the variable. The socket itself and its name are available in *snmp_node* and *snmp_nodename*.

SENDING AND RECEIVING MESSAGES AND DATA

There are three functions for sending control message:

ng_output() sends a control message along the given *path*.

ng_output_node() sends a control message to the node with name *node* and

ng_output_id() sends a control message to the node with node id *id*.

Each of these functions takes the following arguments:

cookie is the node specific command cookie,

opcode is the node specific code for the operation to perform,

arg is a pointer to the message itself. This message must start with a *struct ng_mesg*.

arglen is the overall length of the message (header plus arguments).
The functions return the message id that can be used to match incoming responses or -1 if an error occurs.

Another class of functions is used to send a control message and to wait for a matching response. Note, that this operation blocks the daemon, so use it only if you are sure that the response will happen. There is a maximum timeout that is configurable in the MIB variable *begemotNgTimeout*. Other messages arriving while the functions are waiting for the response are queued and delivered on the next call to the module's idle function.

ng_dialog() sends a control message along the given *path* and waits for a matching response.

ng_dialog_node() sends a control message to the node with name *node* and waits for a matching response.

ng_dialog_id() sends a control message to the node with id *id* and waits for a matching response.

All three functions take the same arguments as the **ng_output***() functions. The functions return the response message in a buffer allocated by malloc(3) or NULL in case of an error. The maximum size of the response buffer can be configured in the variable *begemotNgResBufSiz*.

A data message can be send with the function **ng_send_data**(). This function takes the name of the *snmp_node*'s hook through which to send the data, a pointer to the message buffer and the size of the message. It returns -1 if an error happens.

ASYNCHRONOUS CONTROL AND DATA MESSAGES

A module can register functions to asynchronously receive control and data message.

The function **ng_register_cookie**() registers a control message receive function. If a control message is received, that is not consumed by the dialog functions, the list of registered control message receive functions is scanned. If the cookie in the received message is the same as the *cookie* argument to the **ng_register_cookie**() call and the *id* argument to the **ng_register_cookie**() call was either 0 or equals the node id which sent the control message, the handler function *func* is called with a pointer to the received message, the hook on which the message was received (or NULL if it was received not on a hook), the id of the sender's node and the *uarg* argument of the registration call. The handler function should not modify the contents of the message, because more than one function may be registered to the same cookie and node id.

A control message registration can be undone by calling **ng_unregister_cookie**() with the return value of the registration call. If an error occurs while registering, **ng_register_cookie**() returns NULL.

A module can call **ng_register_hook**() to register a callback for data messages on one of the *snmp_node*'s hooks. If a data message is received on that hook, the callback function *func* is called with the hook name, a pointer to the data message, the size of the message and the argument *uarg* to the registration function. The message should be treated as read-only. A data message registration can be undone by calling **ng_unregister_hook**() with the return value of the registration call. If an error occurs while registering, **ng_register_hook**() returns NULL.

The function **ng_unregister_module**() removes all control and data registrations for that module.

FINDING NODES AND NODE CHARACTERISTICS

The function **ng_node_id**() returns the id of the node addressed by *path* or 0 if the node does not exists.

The function **ng_node_id_node**() returns the id of the node with name *node* or 0 if the node does not exist.

The function **ng_node_node**() retrieves the name of the node with id *id* and writes it to the buffer pointed to by *name*. This buffer should be at least NG_NODESIZ bytes long. The function returns the node id or 0 if the node is not found.

The function **ng_node_type**() retrieves the name of the node with id *id* and writes it to the buffer pointed to by *type*. This buffer should be at least NG_TYPESIZ bytes long. The function returns the node id or 0 if the node is not found.

The function **ng_peer_hook_id**() writes the name of the peer hook of the hook *hook* on the node with *id* to the buffer pointed to by *peer_hook*. The buffer should be at least NG_HOOKSIZ bytes long. The function returns 0 if the node and the hook is found, -1 otherwise. The function skips intermediate tee nodes (see ng_tee(4)).

The function **ng_next_node_id**() returns the node id of the peer node that is on the other side of hook *hook* of node *id*. If *type* is not NULL, the function checks, that the peer node's type is *type*. The function skips intermediate tee nodes (see ng_tee(4)). It returns the node id of the peer node or 0 if an error occurs or the types do not match.

CHANGING THE GRAPH

A number of functions can be used to create or destroy nodes and hooks.

The function **ng_mkpeer_id**() creates a new node of type *type* whose hook *peerhook* will be connected to *hook* of node *id*. If *name* is not NULL the new node is named with this name. The function returns The node id of the new node or 0 if an error happens.

The functions **ng_connect_node**() and **ng_connect_id**() make a new hook connecting *ourhook* of the modules socket node *snmp_node* to *peerhook* of the node identified by id *id* or name *node*. The functions return 0 on success or -1 otherwise.

The function **ng_connect2_id**() connects hook *ourhook* of the node with id *id* to hook *peerhook* of the node with id *peer*. The functions return 0 on success or -1 otherwise. The function **ng_connect2_tee_id**() does the same as **ng_connect2_id**() except, that it puts an unnamed tee node between the two nodes.

The function **ng_rmhook**() removes hook *hook* on the module's *snmp_node*. The function **ng_rmhook_id**() removes hook *hook* on the node with id *id*. The function **ng_rmhook_tee_id**() additionally shuts down all tee nodes between the node and the first non-tee peer.

The function **ng_shutdown_id**() destroys the given node.

FILES

/usr/share/bsnmp/defs/netgraph_tree.def
The description of the MIB tree implemented by **snmp_netgraph**.

/usr/share/bsnmp/mibs/BEGEMOT-NETGRAPH.txt
This is the MIB that is implemented by this module.

SEE ALSO

gensnmptree(1), snmpmod(3)

AUTHORS

Hartmut Brandt <*harti@FreeBSD.org*>