5 Network Device Driver Reference Pages
This chapter contains manual reference pages for the data structures, kernel support routines, and macros essential for HP-UX networking device drivers.

The following data structures are used by the network interface layer:

- **hw_ift**
  - Defined in `/usr/conf/sio/lan_dlpikrn.h`. See `hw_ift(NET4)`.

- **hw_dlpi**
  - Defined in `/usr/conf/sio/lan_dlpikrn.h`. See `hw_dlpi(NET4)`.

Each device driver may maintain its `hw_ift_t` and `hw_dlpi_t` structure as part of a larger structure, the driver control block. The driver control block provides information used in driving and controlling the interface hardware.

The other reference pages describe the routines and macros for use specifically by networking device drivers. Each networking driver may use any or most of the routines and macros on the following reference pages, as well as other routines explained in other sections of this manual.
Network Macros, Structures, and Functions
NAME

ETHER_MAP_IP_MULTICAST – Macro to translate IP multicast address to physical multicast address

SYNOPSIS

#include <netinet/if_ether.h>

void ETHER_MAP_IP_MULTICAST (struct in_addr * ipaddr, u_char enaddr[6]);

PARAMETERS

ipaddr Pointer to the IP multicast address.
enaddr Returned pointer to a 6-byte physical address.

DESCRIPTION

The ETHER_MAP_IP_MULTICAST() network service is a macro that maps an IP multicast address to a 6-byte physical multicast address. It may be used by the Ethernet/802.3 and FDDI protocols. The high-order 25 bits of the physical address are statically assigned. The low-order 23 bits are taken from the low-order bits of the IP address.

RETURN VALUES

None.

CONSTRAINTS

SEE ALSO
NAME

hw_ift (NET4) – Networking commands support structure

SYNOPSIS

```c
#include "/usr/conf/sio/lan_dlpikrn.h"

typedef struct hw_ift
{
    hw_dlpi_t     hp_dlpi;
    u_int         mac_type;
    u_int         llc_flags;
    u_int         mjr_num;
    u_int         nm_id;
    u_int         instance_num;
    u_int         mtu;
    char          *name;
    u_char         hdw_path[MAX_HDW_PATH_LEN];
    u_int         hdw_state;
    u_int         mac_addr_len;
    u_char         mac_addr[MAX_MAC_ADDR_LEN];
    u_int         features;
    /* This is placeholder for future features which 
     * DLPI may support.
     */
    uint8_t       *arpmod_name;
    uint32_t       ppa;
    uint32_t       watch_timer;
    u_int         reserved1;
    /* For Internal use only. */
    lock_t         *hwift_lock;
    /* MP protection. */
    struct hw_ift  *next;
} hw_ift_t;
```

DESCRIPTION

The `hw_ift` network structure provides a consistent method for the network system utilities, `lanscan` and `lanadmin`, to display detailed information on all network devices. (For information on the HP-UX local loopback diagnostic, refer to `loopback (1M)`
The `hw_ift` structure also contains the `hp_dlpi` structure interface for support between HP-UX DLPI and device drivers.

**STRUCTURE**

- **hp_dlpi**: It must be initialized to all zeros by using `bzero()`.

- **mac_type**: A network media device type defined in `/usr/conf/sio/lan_dlpikrn.h`:
  - DEV_8023: IEEE 802.3 device.
  - DEV_8025: IEEE 802.5 device.
  - DEV_ATM: ATM device.
  - DEVETHER: Ethernet device.
  - DEV_FC: Fibre Channel device.
  - DEV_FDDI: FDDI device.

- **llc_flags**: Link Level Control (LLC) encapsulation method. The flag values defined in `/usr/conf/sio/lan_dlpikrn.h` and listed below are used by the `hw_ift` structure to indicate the protocol type and encapsulation method.
  - ETHERTYPE: Ethernet type.
  - IEEE: IEEE 802.2 type.
  - NOVELL: Novell packet type.
  - SNAP: SNAP type.

- **mjr_num**: Major number of the device file associated with the device. The major number could be statically assigned; if it is set to -1 the major number is allocated dynamically during driver initialization.

- **nm_id**: Network management ID. `nm_id` should be initialized via a call to the `get_nmid()` routine.

- **instance_num**: Device instance number. This number is in the value returned by calling the `wsio_isc_to_instance()` routine.

  Note: The `wsio_isc_to_instance()` routine cannot be called from a `driver_attach()` routine, but it can be called from a `driver_init()` routine.
mtu

Maximum transmission unit (MTU) size (number of bytes) for the particular type of link or encapsulation. The following are the MTU values (maximum transmission unit without header) and the defining header files for particular protocols used in type 1 connectionless communication.

<table>
<thead>
<tr>
<th>mtu Value</th>
<th>Header File</th>
<th>Maximum Packet Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETHERMTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>1500 bytes, Ethernet</td>
</tr>
<tr>
<td>FDDI_MTU</td>
<td>&lt;netinet/if_etherinet/if_ether.h&gt;</td>
<td>4352 bytes, SNAP for FDDI</td>
</tr>
<tr>
<td>IEEE8023_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>1497 bytes, IEEE 802.3</td>
</tr>
<tr>
<td>IEEE8025_16_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>4170 bytes, 16 Mb Token Ring</td>
</tr>
<tr>
<td>IEEE8025_4_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>4170 bytes, 4 Mb Token Ring</td>
</tr>
<tr>
<td>SNAP8023_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>1492 bytes, SNAP 802.3</td>
</tr>
<tr>
<td>SNAP8025_16_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>4170 bytes, SNAP for 16 Mb Token Ring</td>
</tr>
<tr>
<td>SNAP8025_4_MTU</td>
<td>&lt;netinet/if_ether.h&gt;</td>
<td>4170 bytes, SNAP for 4 Mb Token Ring</td>
</tr>
</tbody>
</table>

NOTE

The <netinet/if_ether.h> header file was not delivered in Release 10.20. You can obtain a copy of the file by contacting the Interface Program at e-mail address interface@fc.hp.com.

name

Driver device name that is used for naming shared libraries for lanscan and lanadmin.
Hardware path, which can be accessed by calling `io_node_to_hw_path` followed by `io_hw_path_to_str`.

**hdw_state**
Hardware state of the device: 0 if the device is OK. If the device is not available, `hdw_state` must be set to `LAN_DEAD`.

**mac_addr_len**
Number of bytes of `mac_addr` for MAC address.

**mac_addr**
MAC address of the device.

**features**
Features supported by device. Six flags are supported:

- **DRV_MBLK**
  This flag must be set since the third party network driver is purely based on STREAMS model.

- **DRV_MP**
  Set if the device driver is MP-scalable; that is, it uses `spinlock()` and `spinunlock()` to avoid race conditions. With this flag set, the driver cannot use any `splN()` calls.

- **DRV_IP_MULTICAST**
  This flag must be set if driver supports IP multicast feature.

- **DRV_LANC_PROMISC_SUPPORT**
  This flag must be set if driver supports promiscuous listening.

- **DRV_NO_FAST_PATH**
  This flag must be set if driver does not support fast path as described in “Transmission of Message Blocks”.

- **DRV_CKO**
  This flag must be set if driver supports TCP or UDP checksum calculations in hardware.

**arpmod_name**
The name of ARP streams helper module. This helper module complements the generic ARP module to resolve addresses in networks such as Token Ring and Fibre Channel.

**ppa**
PPA number for the interface. The driver should initialize this field with `hw_ift->instance_num`.
watch_timer  For Hewlett-Packard internal use only. This field must be set to 0 for non-Hewlett-Packard devices.

reserved1  Hewlett-Packard internal use only. It must be set to 0 for a non-Hewlett-Packard device.

hwift_lock  Pointer to a hwift_lock spinlock structure to protect the hw_ift structure. It is initialized in hw_ift_attach().

next  Pointer to the next hw_ift structure in the list. This field is set by calling the hw_ift_attach() routine during device driver initialization.

SEE ALSO

bzero(KER2), driver_attach(WSIO_DRV),
driver_if_init(WSIO_DRV), hw_ift_attach(NET3),
<driver>admin(1M), lanscan(1M), loopback(1M), spinlock(KER2),
spinunlock(KER2), wsio_isc_to_instance(WSIO3)
NAME

`hw_ift_attach(NET3)` – Link the `hw_ift` structure to a global list of `hw_ift` structures of active interfaces

SYNOPSIS

```
hw_ift_attach (hw_ift_t *hw_ift_ptr);
```

PARAMETERS

`hw_ift_ptr` Pointer to the associated `hw_ift` structure.

DESCRIPTION

The `hw_ift_attach()` network function links the `hw_ift` structure to a global list of `hw_ift` structures of active interfaces.

It is usually executed in your `driver_init()` routine. It is safely called after the driver claims the ICS: `isc_claim()`

CONSTRAINTS

SEE ALSO

`driver_if_init(WSIO_DRV), hw_ift(NET4)`
NAME

HWIFT_LOCK(NET3), HWIFT_UNLOCK(NET3) – Acquire/release hwift_lock spinlock

SYNOPSIS

#include "/usr/conf/sio/lan_dlpikrn.h"
void HWIFT_LOCK (lock_t lock);
void HWIFT_UNLOCK (lock_t lock);

PARAMETERS

lock Pointer to an allocated lock.

DESCRIPTION

The HWIFT_LOCK() and HWIFT_UNLOCK() network services are macros that acquire and release the hwift_lock spinlock to protect the hw_if field structure fields.

One spinlock, the hwift_lock field in the hw_if structure, is defined to protect the mac_addr and MIB structures pointed to by mib_ptr in the hw_dlpi structure.

NOTE

The hwift_lock spinlock is allocated and initialized by the hw_if_attach() routine. As a result, the HWIFT_LOCK() and HWIFT_UNLOCK() macros are not available until the hw_if_attach() routine has been executed.

Any code that tries to change the hw_if->mac_addr field or the MIB structure pointed to by the hw_dlpi->mib_ptr field should be encapsulated by the HWIFT_LOCK() and HWIFT_UNLOCK() macros. HWIFT_LOCK() must also be used to acquire the hw_if_lock in cases where the dlpi_ioctl() request is not able to complete immediately.

RETURN VALUES

None.
HWIFT_LOCK(NET3), HWIFT_UNLOCK(NET3)

CONSTRAINTS

SEE ALSO

hw_ift(NET4), hw_ift_attach(NET3)
NAME

_kget_log_instance_(NET3) – Produce a unique number for use as a log instance

SYNOPSIS

#include <net_diag.h>

unsigned short kget_log_instance();

PARAMETERS

None.

DESCRIPTION

The _kget_log_instance_() network function returns a unique number tied to the specific instance of a networking interface. This number is for use as a log instance value, for threading log messages together so all messages for the same instance can be identified together. A change in the log instance means that a new event is being logged.

The log instance value should be passed between subsystems through their interface parameter list, so each module may access it. If a module encounters a unique event, it obtains a log instance value. Otherwise, the module uses the current log instance value it was passed without calling _kget_log_instance_.

RETURN VALUES

n A unique number for use as a log instance value.

CONSTRAINTS

SEE ALSO

_klogg_write_(NET3)
NAME

KLOG_CK(NET3) – Checks whether logging is enabled for the current subsystem

SYNOPSIS

```c
#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int KLOG_CK (int subsys_id, int log_class);
```

PARAMETERS

- `subsys_id`: The unique ID number (assigned by Hewlett-Packard) of the calling subsystem.
- `log_class`: Defines the classification of event. All classes are defined in the header file `<sys/subsys_id.h>`. Four classes are defined for logging messages:
  - INFORMATIVE: Normal messages only.
  - WARNING: Warning messages.
  - ERROR: Error condition messages.
  - DISASTER: Critical error messages.

DESCRIPTION

The KLOG_CK() network service is a macro that allows the calling process to find out whether logging is enabled for the current subsystem.

RETURN VALUES

- **0**: Logging is disabled.
- **1**: Logging is enabled.

CONSTRAINTS
SEE ALSO

klogg_write(NET3)
NAME

klogg_write(NET3) – Send log messages to the kernel trace and log facility

SYNOPSIS

#include <net_diag.h>
#include <subsys_id.h>

int klogg_write (short subsys_id, int class, int device_id,
               u_short log_instance, caddr_t tl_packet,
               int tl_packet_cnt);

DESCRIPTION

The klogg_write() network function sends log messages to the kernel trace and log facility. Prefiltering is done at the time of the log call, and unwanted messages are dropped.

PARAMETERS

subsys_id The unique ID (number assigned by Hewlett-Packard) of the calling subsystem.

class The classification of event. All classes are defined in the header file <sys/subsys_id.h>. Four classes are defined for logging messages:

    INFORMATIVE Normal messages only.
    WARNING Warning messages.
    ERROR Error condition messages.
    DISASTER Critical error messages.

device_id The device ID number (for example, if_unit) of the calling subsystem message. If this is a non-applicable parameter, pass in -1.

log_instance A unique static number used to identify the thread of events attending an interface. If this is a non-applicable parameter, pass in -1.
Either a pointer to an mbuf chain or a pointer to a set of iovec structures as determined by tl_packet_cnt. This structure is immediately copied into an mbuf chain owned by the tracing and logging facilities, so the calling routine need not copy the data and then pass a pointer to the data.

If -1, tl_packet points to an mbuf chain. If the value is greater than 0, it is the number of iovec structures (as defined in <sys/uio.h>) that tl_packet points to.

RETURN VALUES

This routine always returns a 0.

CONSTRAINTS

SEE ALSO

KLOG_CK(NET3)
NAME

KTRC_CHECK(NET3) – Check whether tracing is enabled; verify tracing activation on a per-interface-device basis.

SYNOPSIS

#include <sys/net_diag.h>
#include <sys/subsys_id.h>

int KTRC_CHECK (int subsys_id, int trace_kind, int device_id);

PARAMETERS

subsys_id The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).
trace_kind The kind of trace. Available kinds are defined in the <sys/subsys_id.h> header file as follows:
ERROR_TRACE_BIT Error tracing mask
HDR_IN_BIT Inbound header tracing mask
HDR_OUT_BIT Outbound header tracing mask
LOGGING_TRACE_BIT Log call tracing mask
LOOP_BACK_BIT For loopback
PDU_IN_BIT Inbound PDU tracing mask
PDU_OUT_BIT Outbound PDU tracing mask
PROCEDURE_TRACE_BIT Procedure entry/exit trace
PTOP_BIT For point to point
device_id The device ID number (for example, the if_unit value of an ifnet structure). It can be used for filtering on a per-interface basis.
DESCRIPTION

The KTRC_CHECK() network service is a macro that verifies whether tracing is enabled for the current subsystem and device interface.

This macro allows tracing on a per-interface-device basis. It can reduce the impact of tracing on performance where link-level tracing is enabled and the system has more than one interface card installed.

RETURN VALUES

0  Tracing is disabled.
1  Tracing is enabled.

CONSTRAINTS

EXAMPLES

A hypothetical driver named lan2 might use this macro as follows:

```c
if (KTRC_CHECK(NS_LS_LAN0, PDU_OUT_BIT,
                (&lan2_ift_ptr->lancift.is_if)->if_unit))
{
    ktrc_write(...);
}
```

SEE ALSO

ktrc_write(NET3)
NAME

ktrc_write(NET3) – Send trace messages to kernel trace and log facility

SYNOPSIS

```c
#include <net_diag.h>
#include <subsys_diag.h>

int ktrc_write (short subsys_id, u_signed kind, int path_id,
                 int device_id, caddr_t tl_packet,
                 int tl_packet_cnt);
```

DESCRIPTION

The ktrc_write() network function sends trace messages to the kernel trace and log facility. Prefiltering is done at the time of the trace call, and unwanted messages are dropped.

PARAMETERS

- **subsys_id**  
The unique subsystem ID of the calling subsystem (number assigned by Hewlett-Packard).

- **kind**  
The kind of trace. All kinds are defined in the header file <sys/subsys_id.h>. The following are the defined trace kind values. They can be ORed to produce the combination of trace kinds.

  - `ERROR_TRACE_BIT`  
    Error tracing mask

  - `HDR_IN_BIT`  
    In bound header tracing mask

  - `HDR_OUT_BIT`  
    Outbound header tracing mask

  - `LOGGING_TRACE_BIT`  
    Log call tracing mask

  - `LOOP_BACK_BIT`  
    For loopback

  - `PDU_IN_BIT`  
    Inbound PDU tracing mask

  - `PDU_OUT_BIT`  
    Outbound PDU tracing mask
PROCEDURE_TRACE_BIT
Procedure entry/exit trace

PTOP_BIT
For point to point

STATE_TRACE_BIT
State machine tracing mask

path_id
The connection path on the host. If this is a nonapplicable parameter, pass in -1.

device_id
The device ID number (for example, if_unit) of the calling subsystem message. If this is a nonapplicable parameter, pass in -1.

tl_packet
Either a pointer to an mbuf chain or a pointer to a set of iovec structures as determined by tl_packet_cnt. The calling routine will pass a pointer (cast to caddr_t) to an mbuf chain or an iovec structure. This structure is immediately copied into an mbuf chain owned by tracing and logging facilities. Therefore, it is not necessary for the calling routine to copy the data and then pass a pointer to it.

tl_packet_cnt
If -1, then tl_packet points to an mbuf chain. If greater than 0, this is the number of the iovec structure that tl_packet points to.

RETURN VALUES
Always returns a 0.

CONSTRAINTS

SEE ALSO
KTRC_CHECK (NET3)
NAME

set_up_8022(NET3) – Set up global information for the filter and formatting functions

SYNOPSIS

```
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_8022 (u_char * buf_ptr, int len, u_char * dst_addr,
                 u_char * src_addr);
```

DESCRIPTION

The set_up_8022() network function sets up global information used by the filter and formatting functions filter_packet(), format_link_nice(), format_link_raw(), and format_link_terse(). This routine walks through the buffer, pointing to 802.2 data, and copies protocol header information to the appropriate global variables used by the filter and formatters.

Call this routine for each PDU_IN or PDU_OUT trace event.

PARAMETERS

- `buf_ptr` Pointer to the beginning of the 802.2 information. It should not include MAC info.
- `len` Length of the buffer, excluding the MAC header.
- `dst_addr` Pointer to the 6-byte destination MAC address, extracted by local methods from the MAC header.
- `src_addr` Pointer to the 6-byte source MAC address, extracted by local methods from the MAC header.

RETURN VALUES

Always returns 0.
CONSTRAINTS

SEE ALSO

set_up_ether(NET3), set_up_ip(NET3), set_up_link(NET3)
NAME

set_up_ether(NET3) – Set up global information for the filter and formatting functions

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_ether(u_char * buf_ptr, int len, u_char * dst_addr,
                 u_char * src_addr, int ether_type);

DESCRIPTION

The set_up_ether() network function sets up global information used by the filter and formatting functions filter_packet(), format_link_nice(), format_link_raw(), and format_link_terse(). This routine walks through the buffer and copies protocol header information to the appropriate global variables used by the filter and formatter routines. Call this routine for each PDU_IN and PDU_OUT trace event that contains Ethernet packets.

PARAMETERS

buf_ptr Pointer to the beginning of the Ethernet data. It should not include the destination address, source address, or Ethernet type information.

len Length of the buffer, excluding destination, source, and Ethernet type.

dst_addr Pointer to the 6-byte destination MAC address, extracted by local methods from the MAC header.

src_addr Pointer to the 6-byte source MAC address, extracted by local methods from the MAC header.

ether_type Ethernet-type field from the MAC header.
**RETURN VALUES**

Always returns 0.

**CONSTRAINTS**

**SEE ALSO**

`set_up_ether(NET3)`

`set_up_8022(NET3)`, `set_up_ip(NET3)`, `set_up_link(NET3)`
NAME

set_up_ip(NET3) – Set up global information for the filter and formatting functions

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_ip (u_char * buf_ptr, int len);
```

DESCRIPTION

The `set_up_ip()` network function sets up global information used by the filter and formatting functions `filter_packet()`, `format_link_nice()`, `format_link_raw()`, and `format_link_terse()`. It walks through the buffer and copies protocol header information to the appropriate global variables used by the filter and formatters.

NOTE

Most link products should not use this routine. Call this routine only when no link information is available for output formatting (for example, `NS_LOOPBACK`).

PARAMETERS

- `buf_ptr` Pointer to the beginning of the IP information, which should not include MAC information.
- `len` Length of the buffer, excluding the MAC header.

RETURN VALUES

Always returns 0.

CONSTRAINTS
SEE ALSO

set_up_8022(NET3), set_up_ether(NET3), set_up_link(NET3)
NAME

`set_up_link(NET3)` – Set up global information for the link layer only

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int set_up_link (u_char * buf_ptr, int len, u_char * dst_addr,
                 u_char * src_addr);
```

PARAMETERS

- `buf_ptr` Pointer to beginning of the Data Link information. It should not include MAC info. The routine does not currently use this parameter, but it is here for future extensions.
- `len` Length of the buffer, excluding the MAC header.
- `dst_addr` Pointer to the 6-byte destination MAC address, extracted by local methods from the MAC header.
- `src_addr` Pointer to the 6-byte source MAC address, extracted by local methods from the MAC header.

DESCRIPTION

The `set_up_link()` network function sets up global information only for the link layer and does not attempt to extract any upper layer information from the traced packet. It does the minimum setup necessary to use the `format_link_nice()`, `format_link_raw()`, and `format_link_terse()` functions.

NOTE

Use this routine only if the packet being formatted cannot be handled by `set_up_8022()`.
RETURN VALUES
Always returns 0.

CONSTRAINTS

SEE ALSO

set_up_8022(NET3), set_up_ether(NET3), set_up_ip(NET3)
NAME

subsys_N_format(NET_DRV) – Routine to format a single trace or log message from subsystem N.

SYNOPSIS

```
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int subsys_N_format (ss_N_fmt_flag_type flags,
                     char * binary-msg-ptr,
                     char * options-ptr, int msg-cat-fd,
                     int error-fd, int output-file-count,
                     fp_result output-files[],
                     char * time-buffer, int time-buffer-length,
                     int print-op, int user-count,
                     user_acct_result users[], err_num status);
```

DESCRIPTION

The subsys_N_format() network function is provided by the subsystem developer. It can have any unique name. You pass the name to Network Services by including it in a shared library that you specify in the nettlgen.conf configuration file. With this naming notification method, several subsystems can use the same subsys_N_format() function.

At run time, the netfmt command loads the library and calls the routine whenever data from the subsystem is encountered.

The subsys_N_format() network function formats a single trace or log message from the N subsystem. It may discard the message based on filter information in the options file defined by the subsys_N_get_options() function associated with the subsystem.

The successful integration of all subformatters for all subsystems requires that all subformatters follow prescribed guidelines. All subsystems should call tl_header_format1() for each binary message that they will be formatting.
Only the `tl_format_fprintf()`, `tl_format_write()`, and `tl_raw_format()` functions should be used to produce output. Only the file descriptors and file pointers provided by the formatter may be used. No other files may be opened, and no user input may be solicited. No signals should be masked. The subformatter should never call `exit()`. The definitions of the various flags should be carefully followed. Subsystems should try to mirror the behavior of other subsystems as much as possible.

You are not permitted to use `printf()` or `write()` to produce output.

For example, to write messages to standard output, you would use the `tl_format_write()` function with the `output-files[0].fd` file descriptor. To write messages to standard error, you would use the `error-fd` file descriptor.

For simplicity and compatibility with anticipated future growth, use the following mechanism to access the information a subformatter requires. A subsystem can call the `tl_get_parms()` function anywhere within the subformatter to get a pointer to all of the information that a subformatter might need. Use this `tl_get_parms()` mechanism whenever possible, as explained further in `tl_get_parms(NET3)`.

All future parameter changes will be made through the `tl_get_parms()` function. For backwards compatibility, the old parameter list remains the same; but ignore it in favor of using the information returned by `tl_get_parms()`.

**PARAMETERS**

`flags` The type of flags is defined as:

```c
typedef struct {
    unsigned verbosity_bit: 1;
    unsigned console_logging: 1;
    unsigned highlight_bit: 1;
    unsigned nice_mode_bit: 1;
    unsigned terse_mode_bit: 1;
    unsigned terse_link_mode_bit: 1;
    unsigned terse_time_mode_bit: 1;
    unsigned map_to_names_bit: 1;
    unsigned reserved: 24;
} ss_N_fmt_flag_type;
```
verbosity_bit  When this bit is set, a high level of verbosity is selected (the default).

console_logging  This bit is set if console logging is enabled, in which case the subformatter should only call the tl_header_format1() routine and provide very minimal additional information (to be kept to one line).

highlight_bit  If this bit is set (the default) highlighted output is enabled.

nice_mode_bit  This bit is set when nice formatting has been enabled (by default, this bit is not set). Nice formatting is the most descriptive mode of formatting. All possible information should be displayed in this mode of output. Nice mode is not usually used for log messages.

terse_mode_bit  This bit is set when terse formatting has been enabled (by default, this bit is not set). Terse formatting should output only one line of output per trace record. Terse mode is not usually used for log messages.

terse_link_mode_bit  If the terse_mode_bit is set, setting the terse_link_mode_bit should cause the link name to be included in the output.

terse_time_mode_bit  If the terse_mode_bit is set, setting the terse_time_mode_bit should cause the timestamp to be included in the output.
map_to_names_bit
This bit is set (the default) when numeric addresses should be resolved into names whenever possible. For example, an IP address should be displayed as a host name if the map_to_names_bit flag is set.

binary-msg-ptr
A pointer to a buffer that contains the binary trace/log message to be formatted. The buffer contains the trace/log header, struct tl_msg_hdr, from the <ntl.h> file, followed by the trace/log data (from ktrc_write() or klogg_write()):

```c
typedef struct {
    unsigned short hdr_len;
    short subsystemid;
    int device_id;
    tl_msg_flag_type flags;
    set_of_32 kind;
    set_of_32 class;
    set_of_32 version;
    int dropped_events;
    unsigned int dropped_data;
    unsigned int data_len;
    unsigned int orig_data_len;
    struct timeval time;
    int invoke_id;
    int path_id;
    unsigned short log_instance;
    short uid;
    unsigned int connection_id;
} tl_msg_hdr_type;
```

For tracing, the data may be truncated by the nettl command facilities. Check the tl_msg_hdr->data_len field to find out how much data was captured.
options-ptr  A pointer to a data structure defined by the subsystem for communication between the 
subs委书记_N_get_options() routine and the 
subs委书记_N_format() routine. If no options are used, 
this pointer is NULL. The actual type of the structure 
pointed to by options-ptr is entirely up to the 
subsystem developer.

msg-cat-fd  The file descriptor of the subsystem message catalog 
configured in nettlgen.conf. The formatter opens 
subsystem message catalogs using catopen(). The 
tl_check_cat_version() function can be used to 
check that the version of the message catalog 
corresponds to the version of the subformatter. 
Subsystems should not open their own message catalog 
files.

error-fd  A file descriptor that identifies the file that receives 
any fatal or nonfatal error messages (typically 
associated with stderr).

output-file-count  The number of output files to receive the formatted 
trace/log messages. It must be 1 for HP-UX.

output-files[]  An array of structures, each of which contains a file 
descriptor number, fd, and a result.

typedef struct 
{
   int fd;
   int result;
} fp_result;

The fd-designated output file, output-files[0].fd, 
receives the formatted trace/log messages. Only one 
output file is used for HP-UX; output-files[0].result 
is ignored. This output file will have been opened by 
the formatter driver.

time-buffer  A string containing the formatted time stamp from the 
trace/log header.

time-buffer-length  Length of the time-buffer string, not counting the null 
terminator.
print-op

For HP-UX, this parameter must be 0.

user-count

For HP-UX, this parameter must be 0.

users

For HP-UX, this parameter must be NULL.

status

Contains an error code value if the routine returns -1, indicating an error condition. `<fmt.h>` gives a complete list of such error codes:

- **FMTERR_INI_FLAGS**
  Invalid flags parameter.

- **FMTERR_INV_BIN_MP**
  Invalid binary-msg-ptr binary message pointer.

- **FMTERR_INV_OUT_FP**
  Invalid output file pointer.

- **FMTERR_INV_MC_FP**
  Invalid message catalog file pointer.

- **FMTERR_INV_TL_MSG**
  Invalid trace/log message. The message is so corrupted that no formatting can be done.

- **FMTERR_SYS_ERROR**
  An error has been returned from a system call.

**RETURN VALUES**

- **0**
  Successful completion.

- **-1**
  Error. An appropriate error code is provided in the status field and an error message (if any) is given in the file named in `error-fd`.

**CONSTRAINTS**
SEE ALSO

catopen(3C), exit(2), klogg_write(NET3), ktrc_write(NET3),
subsys_N_get_options(NET_DRV), tl_check_cat_version(NET3),
tl_format_fprintf(NET3), tl_format_write(NET3),
tl_get_Parms(NET3), tl_header_format1(NET3),
tl_raw_format(NET3)
NAME

subsys_N_get_options(NET_DRV) – Routine to process options for subsystem N

SYNOPSIS

#include <fmt.h>

int subsys_N_get_options (get_opt_parms_type * get-opt-parms-ptr);

DESCRIPTION

The subsys_N_format() network function is provided by the subsystem developer. It can have any unique name. You pass the name to Network Services by including it in a shared library that you specify in the nettlgen.conf configuration file. With this naming notification method, several subsystems can use the same subsys_N_get_options() function.

The subsys_N_format() network function is provided by the subsystem developer. It can have any unique name. You pass the name to Network Services by including it in a shared library that you specify in the nettlgen.conf configuration file. With this naming notification method, several subsystems can use the same subsys_N_get_options() function.

It is the responsibility of the subsys_N_get_options() routine to read the subsystem-specific options information from the filter command file and store any necessary information.

PARAMETERS

get-opt-parms-ptr

Pointer to a get_opt_parms_type structure that describes the subsystem.
STRUCTURE

The `get_opt_parms_type` structure is defined in `<fmt.h>` as:

```c
typedef struct {
    int *status_ptr;
    FILE *subsys_strm;
    FILE *error_strm;
    FILE *log_strm;
    int ss_id;
    char *ss_name;
    nl_catd ss_msg_cat;
    get_opt_flag_type ss_n_get_opt_flag;
    char **ss_options_ptr_ptr;
    int ss_output_fd;
    char *options_file_name;
} get_opt_parms_type;
```

`status_ptr` The error code indicating an error condition of the routine if the returned value is -1; `<fmt.h>` gives a complete list of such error codes:

- **FMTERR_INV_FLAGS** Invalid `ss_n_get_opt_flag` parameter.
- **FMTERR_NO_MEM** There is not enough memory; a call to `malloc()` failed.
- **FMTERR_INV_OPT_FD** Invalid options file descriptor passed in.
- **FMT_INV_MC_FD** Invalid message catalog file descriptor passed in.
- **FMTERR_SYS_ERROR** An error has been returned from a system call.
subsys_strm A pointer to the file that refers to the temporary file containing the options specifically for the N subsystem. This file is created by the caller prior to invoking the subsys_N_options() routine, and each line has been converted to lowercase. All comments, blank lines, and lines for other subsystems are already removed. In addition, the keyword identifying this subsystem has been stripped off each line, so only the options for this particular subsystem are in the file. Due to a special encoding of line number and other data, the tl_get_line() routine must be used to get option lines from this stream file.

error_strm A pointer to the file that is to receive error messages.

log_strm A pointer to the file that is to receive a summary of all options and files in effect for the subsystem, generated by the subsys_N_get_options() routine. The nettl command reports the contents of this file after all the subsystems have finished reading their respective filter command files.

ss_id The subsystem ID number for the subsystem as found in the configuration file.

ss_name The subsystem name for the subsystem as found in the configuration file.

ss_msg_cat A file descriptor pointing to the message catalog for the subsystem as found in the configuration file.

ss_n_get_opt_flag An option flag, defined as:

```c
typedef struct
{
    u_int trace_log_bit: 1;
    u_int parse_only_bit: 1;
    u_int reserved: 30;
} get_opt_flag_type;
```

trace_log_bit This flag is not needed and should not be used by subsys_N_get_options.
parse_only_bit

The flag is set when the subsys_N_get_options() routine does not need to process the information in the file, but only parse the input and check for syntax and semantic errors.

ss_options_ptr_ptr

A pointer to a pointer to a data structure containing the specific information processed by subsys_N_get_options() and passed on to subsys_N_format() to handle special formatting. This structure should be allocated and initialized by subsys_N_get_options().

ss_output_fd

A file descriptor referring to the file receiving the formatter output.

options_file_name

A character string which contains the file name of the filter file passed to netfmt with the -c option. The file name can be used in error and warning messages produced by subsystem_N_get_options() while parsing the filter field. Subsequent messages need not display the file name.

RETURN VALUES

0 Successful completion.

-1 Error. An appropriate error code is provided in the status_ptr field and an error message (if any) is given in the file named in error-strm.

CONSTRAINTS

SEE ALSO

malloc(3C), nettl(1M), subsys_N_format(NET_DRV), tl_get_line(NET3)
NAME

tl_banner_char(NET3) – Get the character used for tl_header_format1() banner printing

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

char tl_banner_char(unsigned int kind_class);
```

PARAMETERS

`kind_class` The trace kind or log class of the message.

DESCRIPTION

The `tl_banner_char()` network function obtains the character to be used when printing a header banner with the `tl_header_format1()` function. The character is based on the type of log class or trace kind. This function helps to ensure consistent banners for all trace/log messages.

RETURN VALUES

`tl_banner_char()` always returns a character to be used by `tl_header_format1()`.

CONSTRAINTS

SEE ALSO

`tl_header_format1(NET3)`
NAME

tl_check_cat_version(NET3) – Check compatibility between subsystem message catalog and subsystem formatter library

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int tl_check_cat_version (int msgcatfd, int setnum,
                         int msgnum, char *expectedversion,
                         FILE *errstream);

PARAMETERS

msgcatfd A file descriptor of the message catalog which contains the version string.
setnum The set number in the message catalog.
msgnum The message number in the message catalog.
expectedversion The version string that the message catalog is expected to contain.
errstream A FILE pointer to a stream that will receive error messages.

DESCRIPTION

The tl_check_cat_version() network function checks that the subsystem message catalog has a compatible version with the subsystem formatter library. If not, an appropriate warning message is issued.

RETURN VALUES

0 Successful completion. The versions match.
-1 Error. The versions don’t match or the file descriptor of the message catalog is invalid.
CONSTRAINTS

SEE ALSO

\texttt{tl\_header\_format1(NET3)}
NAME

`tl_format_fprintf(NET3)` – Convert, format, and print arguments to standard output

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int tl_format_fprintf(FILE *stream, format_wrt_flag_type flags,
                       error_num *status_ptr,
                       char *format, [, arg...]);
```

PARAMETERS

- **stream**
  One of the FILE streams contained in the `ss_N_fmt_parms_type` structure returned by `tl_get_parms()`.
- **flags**
  Controls the output behavior of the `tl_format_fprintf()` routine. The value must be set before calling `tl_format_fprintf()`.

```
typedef struct
{
  unsigned highlight   : 1;
  unsigned wait_to_write : 1;
  unsigned reserved    : 30;
} format_wrt_flag_type;
```

- **highlight**
  Write the format data in inverse video.
- **wait_to_write**
  Reserved for future use.
- **status_ptr**
  Contains the error value if the routine returns a -1.
format

The format character string contains two types of objects: plain characters that are copied to the output stream, and conversion specifications. Each string results in fetching 0 or more arguments, arg. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args are ignored.

arg

Argument for the format character string.

DESCRIPTION

The `tl_format_fprintf()` network function converts, formats, and prints its arguments under control of the format. This routine behaves like `printf()` but must be used instead to give `netfmt()` control over the formatted buffer.

RETURN VALUES

0 Successful completion.
-1 Error.

Fatal errors are reported through the return value and the `status_ptr` parameter. All error messages (as follows) are written to the file pointed to by the `error_fd` parameter of the `subsys_N_format()` routine.

FMTERR_FORMAT_FPRINTF An error occurred in writing to the output stream.

FMTERR_INV_L_STR Invalid line pointer string.

FMTERR_INV_OUT_FD Invalid output file descriptor.

FMTERR_SYS_ERROR An error has been returned from a system call within the `tl_format_write()` routine.

CONSTRAINTS
Network Device Driver Reference Pages

tl_format_fprintf(NET3)

SEE ALSO

subsys_N_format(NET_DRV), tl_format_write(NET3),
        tl_getParms(NET3), tl_raw_format(NET3)
NAME

`tl_format_write` (NET3) – Write a buffer to standard output

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int tl_format_write (u_char *input_line_ptr,
                     int input_line_byte_count, int error_fd,
                     fmt_wrt_flag_type flags, int output_file_count,
                     fd_result output_files[], int print_op,
                     int user_count, user_acct_result users[],
                     err_num *status_ptr);
```

PARAMETERS

- **input_line_ptr** A character string that contains the message to be printed to the output files. `input_line_ptr` need not be null-terminated nor ended with a newline.
- **input_line_byte_count** The byte count of the `input_line_ptr` message string.
- **error_fd** A file descriptor pointing to a file to receive error messages from the `tl_format_write()` routine.
- **flags** Controls output behavior of the `tl_format_write()` routine. The value must be set before calling `tl_format_write()`.

```c
typedef struct {
    unsigned int highlight : 1;
    unsigned int wait_to_write : 1;
    unsigned int reserved : 30;
} fmt_wrt_flag_type;
```

- **highlight** Write the `input_line_ptr` data in inverse video.
- **wait_to_write** Reserved for future use.
**reserved**

Reserved for future use.

**output_file_count**

This is the number of output files to receive the formatted trace/log header output. For HP-UX, only one output file is used, and the value is always 1.

**output_files[]**

An array of structures consisting of a file descriptor and result variable for each file to receive the formatted trace/log header output. For HP-UX, only one output file is used; output_file[0].fd refers to the file receiving the formatter output.

**print_op**

For HP-UX, this parameter must have a value of 0.

**user_count**

For HP-UX, this parameter must have a value of 0.

**users**

For HP-UX, this parameter must have a value of NULL.

**status_ptr**

Contains the error value if the return value is -1.

**DESCRIPTION**

The `tl_format_write()` network function writes a buffer to standard output, pointed to by `output_file[0].fd`. The buffer may be created by one or more calls to the `printf()` C library function.

**RETURN VALUES**

0  Successful completion.

-1  Error.

Fatal errors are reported through the return value and the `status_ptr` parameter. All error messages (as follows) are written to the file pointed to by `error_fd`:

- `FMTERR_FORMAT_WRITE` An error has occurred in writing to an output file.

- `FMTERR_INV_L_STR` Invalid line pointer string.

- `FMTERR_INV_OUT_FD` Invalid output file descriptor.
FMTERR_SYS_ERROR

An error has been returned from a system call within the `tl_format_write()` routine.

CONSTRAINTS

SEE ALSO

`tl_format_fprintf(NET3), tl_raw_format(NET3)`
NAME

tl_get_line(NET3) – Obtain a line from a filter command file

SYNOPSIS

```c
int tl_get_line (FILE * commandstream, char * line, int linesize,
    char * origline, int lineno, FILE * errorstream);
```

PARAMETERS

- `commandstream`: A FILE pointer that points to the temporary filter command file containing a single subsystem's filter commands; typically the subsys_strm field of the get_optParms_type parameter to subsys_N_get_options().
- `line`: The buffer where `tl_get_line()` stores the filter command line.
- `linesize`: Size of line (no more than 2048).
- `origline`: `tl_get_line()` routine will store the original filter command line as it appeared in the filter command file in `origline`.
- `lineno`: `tl_get_line()` will store the line number of `origline` in `lineno` as it appeared in the filter command file.
- `errorstream`: A FILE pointer to a stream that will receive error messages.

DESCRIPTION

The `tl_get_line()` network function obtains a line from a filter command file.

The core formatter reads the filter command file, collects the lines specific to a subsystem, then edits and stores them into a temporary file. It then calls your `subsys_N_get_options()` routine with a parameter set as a pointer to this temporary file. The `subsys_N_get_options()` routine must call the `tl_get_line()` routine to extract one line at a time from this temporary file for processing. The lines are upshifted, and redundant white space is removed. `tl_get_line()` will store the "cleaned" filter command line in this buffer.
RETURN VALUES

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>0</td>
<td>End of file.</td>
</tr>
<tr>
<td>&lt;0</td>
<td>Error.</td>
</tr>
</tbody>
</table>

CONSTRAINTS

SEE ALSO

subsys_N_get_options(NET_DRV)
NAME

`tl_getParms(NET3)` – Return a pointer to a `ss_N_fmt_parms_type` data structure

SYNOPSIS

```c
#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

ss_N_fmt_parms_type * tl_getParms()
```

DESCRIPTION

The `tl_getParms()` network function returns a pointer to a `ss_N_fmt_parms_type` data structure containing members that a subsystem subformatter needs in order to operate. The core formatter builds and initializes this data structure before calling `subsys_N_format()`.

STRUCTURE

```c
typedef struct
{
    int *ss_status_ptr;
    FILE *ss_output_strm;
    int ss_output_fd;
    FILE *ss_error_strm;
    int ss_error_fd;
    nl_catd ss_msg_cat;
    char *ss_name;
    char *ss_binary_msg_ptr;
    char *ss_options_ptr;
    ss_N_fmt_flag_type ss_n_fmt_flags;
    char *time_buffer;
    int time_buffer_length;
    int output_file_count;
    fd_result output_files[1];
    int print_op;
    int user_count;
    user_acct_result *users;
} ss_N_fmt_parms_type;
```

`ss_status_ptr` Used by a subformatter to store an error code if it fails.
ss_output_strm  A FILE pointer that will receive the formatted trace/log message.

ss_output_fd   A file descriptor that will receive the formatted trace/log messages.

ss_error_strm  A FILE pointer that will receive any fatal or nonfatal error messages.

ss_error_fd    A file descriptor that will receive any fatal or nonfatal error messages.

ss_msg_cat     The subsystems' message catalog descriptor to be used in catgets.

ss_name        A pointer to the subsystem name.

ss_binary_msg_ptr
A pointer to a buffer containing log/trace messages to be formatted.

ss_options_ptr A pointer to a buffer containing information to be passed between the subsys_N_format() routine and the subsys_N_get_options() routine. See the options-ptr parameter in the subsys_N_format() routine.

ss_n_fmt_flags Options flags: the ss_N_fmt_parms_type type is defined in <fmt.h> as:

typedef struct
{
    unsigned verbosity_bit: 1;
    unsigned console_logging: 1;
    unsigned highlight_bit: 1;
    unsigned nice_mode_bit: 1;
    unsigned terse_mode_bit: 1;
    unsigned terse_link_mode_bit: 1;
    unsigned terse_time_mode_bit: 1;
    unsigned map_to_names_bit: 1;
    unsigned reserved: 24;
} ss_N_fmt_flag_type;

verbosity_bit   Set for verbose output.
console_logging Set for console formatting.
highlight_bit   Set for highlighted output.
nice_mode_bit   Set for nice output.
tl_get_parms(NET3)

**terse_mode_bit**  Set for terse output.

**terse_link_mode_bit**  Set for link messages in terse mode.

**terse_time_mode_bit**  Set for timestamp in terse mode.

**map_to_names_bit**  Reserved

**time_buffer**  A string containing the formatted timestamp from the trace/log header (see the time-buffer parameter of subsys_N_format()).

**time_buffer_length**  Length of the time_buffer string, not counting the null terminator.

**output_file_count**  The number of output files to receive the formatted trace/log messages. For HP-UX, this member must have a value of 1.

**output_files[]**  An array of structures consisting of a file descriptor and result variable for each file to receive the formatted trace/log output. For HP-UX, only output_file[0].fd refers to the file receiving the formatter output.

**print_op**  For HP-UX, this member has a value of 0.

**user_count**  For HP-UX, this member has a value of 0.

**users**  For HP-UX, this member has a value of NULL.

**RETURN VALUES**

\textit{n}  Successful completion. The value is a pointer to an ss_N_fmt_parms_type data structure containing members that a subsystem subformatter needs to operate.

**CONSTRAINTS**
SEE ALSO

subsys_N_format(NET_DRV)
tl_header_format1(NET3) – Routine to format a single trace or log header

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int tl_header_format1 (char *header_ptr, int error_fd,
                    ss_N_fmt_flag_type flags,
                    char *kind_str, char banner_char,
                    int output_file_count,
                    fd_result output_files[],
                    char *time_buffer, int time_buffer_length,
                    int print_op, int user_count,
                    user_acct_result users[], int location,
                    err_num *status_ptr);

PARAMETERS

header_ptr  Points to a buffer that contains the header of the trace/log message to be formatted.

error_fd    A file descriptor that refers to the file that will receive any error messages.

flags       Option flags in an ss_N_fmt_flag_type structure, defined in <fmt.h> and described in
            tl_get_parms(NET3).

kind_str    A text message (typically the result of the tl_log_class() or tl_trace_kind() function) to be displayed for the kind field from the trace/log header, This string must be null-terminated. The kind message is truncated to 16 characters. If kind_str is NULL, the kind field from the header is displayed as a decimal value.
**banner_char**  The character to use in the banner header line (typically the result of the `tl_banner_char()` function). The subformatter may use this character to indicate differences in messages, such as inbound or outbound messages. For example, inbound messages could use the character "v" while outbound messages could use the character "^".

**output_file_count**  This is the number of output files to receive the formatted trace/log header output. For HP-UX, only one output file is used so this value is always 1.

**output_files[]**  An array of structures consisting of a file descriptor and a result variable for each file to receive the formatted trace/log header output. For HP-UX, only one output file is used: `output_file[0].fd`.

**time_buffer**  A string depicting the formatted time stamp from the trace/log header.

**time_buffer_length**  The length of `time_buffer` not counting the null terminator byte.

**print_op**  For HP-UX, this parameter must have a value of 0.

**user_count**  For HP-UX, this parameter must have a value of 0.

**users**  For HP-UX, this parameter must have a value of `NULL`.

**location**  A value that can be used to locate the source of the message in the code. This parameter is set by the subsystem and may be used to represent any information the subsystem desires.

**status_ptr**  The error value if the routine returns a -1:

- `FMTERR_INV_HDR`: The trace/log header is invalid (corrupt).
- `FMTERR_INV_HDR_PTR`: The trace/log header pointer is invalid.
- `FMTERR_INV_OUT_FD`: The output file descriptor is invalid.
FMTERR_INV_MC_FD
The message catalog descriptor is invalid.

FMTERR_SYS_ERROR
An error was returned from a system call within tl_header_format1().

DESCRIPTION
The `tl_header_format1()` network function formats a single trace or log header. The format of the output conforms to the standard HP-UX network tracing and logging recommendations. The formatted header is written to the output file specified by `output_file[0]. fd`. `tl_header_format1()` must be called by every subformatter after the subsystem filters have been processed. At a minimum, this may be the only output generated by the subformatter.

RETURN VALUES
0  Successful completion.
-1  Error. Fatal errors are reported through the status_ptr parameter.

All error messages are written to the file pointed to by `error_fd` parameter.

CONSTRAINTS

SEE ALSO
`tl_banner_char(NET3), tl_get_parms(NET3), tl_log_class(NET3), tl_trace_kind(NET3)`
NAME

tl_log_class(NET3) – Return a text interpretation for a log class value

SYNOPSIS

char * tl_log_class(unsigned int class);

PARAMETERS

class The numeric log class of the message. The keywords are defined in <sys/subsys_id.h>.

<table>
<thead>
<tr>
<th>class</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INFORMATIVE</td>
</tr>
<tr>
<td>2</td>
<td>WARNING</td>
</tr>
<tr>
<td>4</td>
<td>ERROR</td>
</tr>
<tr>
<td>8</td>
<td>DISASTER</td>
</tr>
</tbody>
</table>

DESCRIPTION

The tl_log_class() network function returns a text interpretation of a log class. The log class is stored as an integer. This function converts that number into a string that can be used in the formatted output. For example, passing in a log class of 8 causes the return value to be DISASTER. The result of tl_log_class() is typically used as a parameter to tl_header_format1() when printing a header.

RETURN VALUES

<> NULL A pointer to a text interpretation of a log class.
NULL class is not a defined value.

CONSTRAINTS
SEE ALSO

tl_trace_kind(NET3)
NAME

tl_raw_format(NET3) – Format trace or log message into hexadecimal and printable ASCII characters

SYNOPSIS

#include <fmt.h>
#include <ntl.h>
#include <subsys_id.h>

int tl_raw_format (char * data_ptr, int num_bytes,
                    int start, int error_fd, raw_fmt_flag_type flags,
                    int output_file_count,
                    fd_result output_files[],
                    int print_op, int user_count,
                    user_acct_result users[], err_num * status_ptr);

PARAMETERS

data_ptr    A pointer to the buffer that contains the data to be dumped in hexadecimal form.
num_bytes   The number of bytes to dump from the buffer pointed to by data_ptr. There is no checking to ensure that the number of bytes given does not exceed the actual buffer length. If num_bytes is zero, no data will be dumped.
start       The offset into the buffer pointed to by data_ptr where the dump should begin. If start is zero, the dump will begin at the byte pointed at by data_ptr.
error_fd    A file descriptor that will receive error messages.
flags       Reserved for future used; should be set to 0 by the caller.
output_file_count The number of output files to receive the raw dump. For HP-UX, this parameter must have a value of 1.
**DESCRIPTION**

The `tl_raw_format()` network function formats a trace or log message into both hexadecimal and printable ASCII characters. The raw formatted output appears as follows:

```
0:73 61 6d 70 6c 65 5f 6c 6f 67 5f 64 61 74 61 2e sample_log_data
16:20 6d 6f 72 65 5f 64 61 74 61 20 61 73 64 66 6a more_data asdfj
```

The left-most column gives the decimal byte offset. The center area is the hexadecimal display of the data. The right-most column is the printable ASCII display of the data. A period is displayed for any nonprinting character.

**RETURN VALUES**

- **0**  Successful completion.
- **-1**  Error. Fatal errors are reported through the `status_ptr` parameter.

All error messages are written to the file pointed to by the `error_fd` parameter.

**CONSTRAINTS**

**SEE ALSO**

`tl_format_fprintf(NET3), tl_format_write(NET3)`
NAME

tl_trace_kind(NET3) – Return a text interpretation for a trace kind value

SYNOPSIS

#include <sys/subsys_id.h>

char * tl_trace_kind (unsigned int kind);

PARAMETERS

kind  The numeric trace kind of the message. The keywords are defined in <sys/subsys_id.h>.

kind Value and Keyword Description
0x80000000 HDR_IN_BIT Inbound Protocol Header.
0x40000000 HDR_OUT_BIT Outbound Protocol Header.
0x20000000 PDU_IN_BIT Inbound Protocol Data Unit (including header and data).
0x10000000 PDU_OUT_BIT Outbound Protocol Data Unit (including header and data).
0x08000000 PROCEDURE_TRACE_BIT Procedure entry and exit.
0x04000000 STATE_TRACE_BIT Protocol or connection states.
0x02000000 ERROR_TRACE_BIT Invalid events or condition.
0x01000000 LOGGING_TRACE_BIT Special kind of trace that contains a log message.
0x00800000 LOOP_BACK_BIT Packets whose source and destination system are the same.
0x00400000 PTOP_BIT Packets whose transmission is point to point.
DESCRIPTION

The `tl_trace_kind()` network function returns a text interpretation of a trace kind. The trace kind is stored as an integer. This function converts that number into a string that can be used in the formatted output. For example, passing in a trace kind of 0x80000000 causes the return value to be "HDR IN TRACE". The result of `tl_trace_kind()` is typically used as a parameter to `tl_header_format1()` when printing a header.

RETURN VALUES

- `<NULL` A pointer to a text interpretation of a trace kind.
- `NULL` kind is not a defined value.

CONSTRAINTS

SEE ALSO

- `tl_log_class(NET3)`