ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Platform Integration
Developer Guide

December 2015

IP Infusion Inc. Proprietary

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

Contents

Preface

This guide describes the components that are used to integrate ZebOS-XP on different platforms.

Audience

This guide is intended for developers who write code to port ZebOS-XP to different platforms.

Conventions

Table P-1 shows the conventions used in this guide.

Table P-1: Conventions

Convention	Description
Italics	Emphasized terms; titles of books
Note:	Special instructions, suggestions, or warnings
`monospaced type`	Code elements such as commands, functions, parameters, files, and directories

Contents

This guide contains these chapters:

- Chapter 1, *Introduction*
- Chapter 2, *Hardware Abstraction Layer*
- Chapter 3, *Hardware Services Layer*
- Chapter 4, *Platform Abstraction Layer*
- Chapter 5, *System Messages*

Related Documents

The following guide is related to this document:

- *Architecture Guide*

Note: All ZebOS-XP technical manuals are available to licensed customers at http://www.ipinfusion.com/support/document_list.

Support

For support-related questions, contact support@ipinfusion.com.

Comments

If you have comments, or need to report a problem with the content, contact techpubs@ipinfusion.com.

CHAPTER 1 Introduction

This chapter provides an overview of how ZebOS-XP integrates with various platforms. It also includes an overview of the features of platform integration and describes the system architecture.

ZebOS-XP can be integrated on a variety of platforms. This integration allows you to rapidly configure full metro Ethernet switching and traffic management solutions without costly software and hardware development that is associated with complex device development.

The ZebOS-XP platform integration feature use the following components:

- Hardware Abstraction Layer (HAL): Control plane interface to communicate with the hardware. See Chapter 2, *Hardware Abstraction Layer* for more about HAL.
- Hardware Services Layer (HSL): Interface for the hardware and the operating system (OS). It communicates directly with the control plane. See Chapter 3, *Hardware Services Layer* for more about HSL.
- Platform Abstraction Layer (HAL): Communicates with the operating system for forwarding table updates. See Chapter 4, *Platform Abstraction Layer* for more about PAL.
- Socket Communication Layer: Communicates between the control plane and the forwarding plane. See Chapter 2, *Hardware Abstraction Layer* for more about the Socket Communication Layer.

Figure 1-1 shows the ZebOS-XP system architecture.

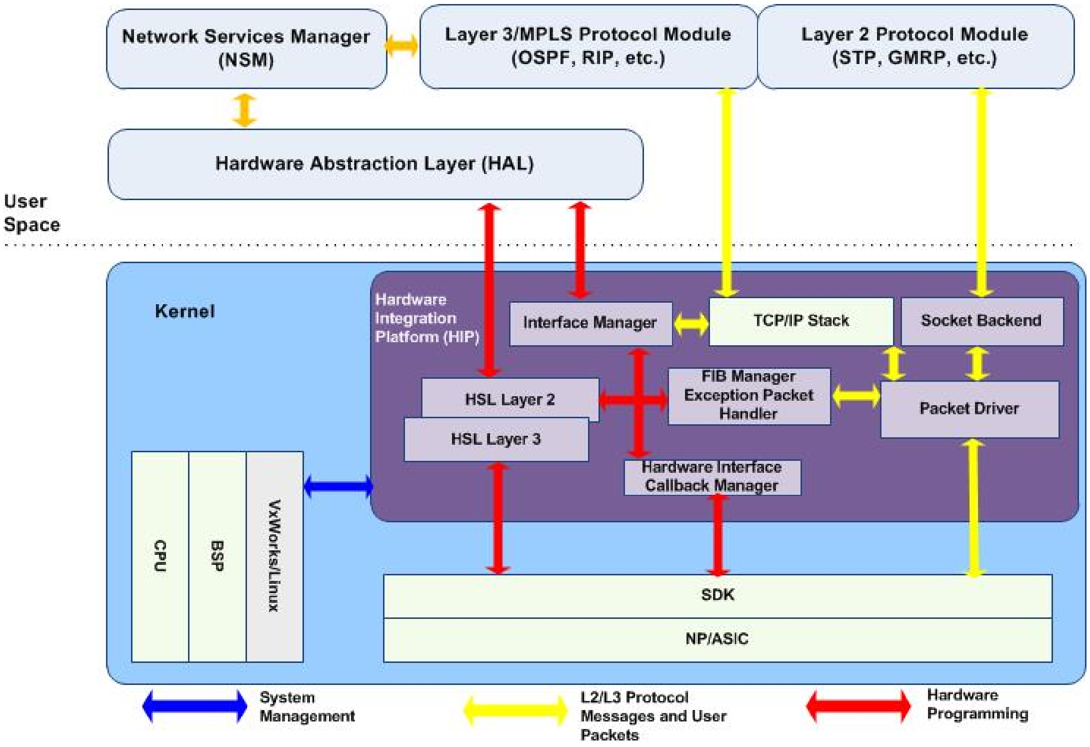
Network Services Manager (NSM)
Layer 3/MPLS Protocol Module (OSPF, RIP, etc.)
Layer 2 Protocol Module (STP, GMRP, etc.)
Hardware Abstraction Layer (HAL)
User Space
Kernel
Hardware Integration Platform (HIP)
Interface Manager
TCP/IP Stack
Socket Backend
HSL Layer 2
HSL Layer 3
FIB Manager Exception Packet Handler
Packet Driver
Hardware Interface Callback Manager
CPU
BSP
VxWorks/Linux
SDK
NP/ASIC
System Management
L2/L3 Protocol Messages and User Packets
Hardware Programming

Figure 1-1: ZebOS-XP system architecture

CHAPTER 2 Hardware Abstraction Layer

This chapter provides information about the Hardware Abstraction Layer (HAL). It includes an overview, list of features and all of the relevant ACL functions.

Overview

HAL module enables the control plane to run on different hardware platforms, regardless of the type of chip set or operating system being used. Additionally, it makes the control plane fully independent of the specific TCP/IP stack implementation being used with the operating system. HAL has two major functions:

- **Encapsulating and sending control messages:** These messages originate from the control module (NSM)
- **Processing System Responses**: HAL processes system responses and events, and calls the appropriate module in the control plane based on the type of event. The processing of system layer responses includes parsing or decoding of the message, and then notifying the appropriate module in the control plane, if required

Socket Communication Layer

HAL communicates with the Hardware Services Layer (HSL) via the socket communication layer. This layer communicates with the control plane and the forwarding plane. It provides the following communication tasks:

- **Programming Message Delivery**. A command interface is responsible for the delivery of configuration programming messages for the hardware and the operating system.
- **Socket Communication Layer**. An event interface manages the messages initiated by the forwarding plane.
- **Polling**. A polling interface gets information about interfaces.

Communication is based on a client/server model. The HSL forwarding plane implements the server side; the control plane implements the client side. During hardware initialization, the forwarding plane creates a server socket to which control plane clients connect once they start. Communication between the control plane and the forwarding plane is message-based. Each message contains the following:

- Message Type
- Message Length
- Message Data in the TLV encoded (length/field) format

The sender is responsible for preparing the message header, and encoding the data. The receiver is responsible for decoding messages, acknowledging receipt of the message to the sender, and performing an integrity check.

Socket Mechanisms

ZebOS-XP provides one proprietary socket mechanism, the full AF_HSL family socket implementation, which is registered with the OS. It serves as the best performance communication interface.

Data Structures

The following subsection list the data structures for HAL. The following two data structures are used by multiple ZebOS-XP modules and are documented in the *Common Data Structures Developer Guide*:

- interface
- lib_globals

hal_apbf_acl_id_status

This structure is the HAL structure for ACL filter status.

Type	Definition
packet_count	The number of packets that hit a rule qualifier. This is retrieved from the hardware.

Definition

```
struct hal_apbf_acl_id_status
{
  ut_int64_t packet_count;    /* Number of packets which hit the qualifier for
                                 a rule. To be retrieved from hw. */
};
```

hal_apbf_rule_status

This structure is used to pass the rule to get the status of the ACL filter.

Type	Definition
apbf_flag	APBF flags.
num_ifindices	Number of interface indices.
ifindices	Interface indices.
rule_group_id	Rule group identifiers.
rule_id	Rule identifiers.
acl_id	ACL identifiers.

Definition

```
/* */
struct hal_apbf_rule_status
{
  u_int8_t apbf_flag;
  u_int32_t num_ifindices;
  u_int32_t *ifindices;
  u_int32_t rule_group_id;
  u_int32_t rule_id;
  u_int32_t acl_id;
};
```

hal_msg_apbf_rule

This structure is for APBF HAL messages.

Type	Definition
rule_group_id	Rule group identifier.
total_rule_entries	Number of rule entries. That is, the number of rules and the number of ACL filters.
total_rule_delta_entries	Number of rule entries. That is, the number of rules and the number of ACL filters.
num_rule_entries	Number of rule entries being sent to the hardware socket layer (HSL) in a call.
num_rule_delta_entries	Number of rule entries being sent to the hardware socket layer (HSL) in a call.
num_ifindices	Number of interfaces that this rule should be applied.
ifindices	Array of interface indices.
hal_apbf_rule_list_delta	List of APBF rules for a rule group that are being added or deleted. Updates are part of the “struct hal_apbf_rule” type.
hal_apbf_rule	List of APBF rules for a rule group that are set by the “struct hal_apbf_rule rule type.

Definition

```
struct hal_msg_apbf_rule
{
  u_int32_t rule_group_id;           /* Rule Group Identifier */
  u_int32_t total_rule_entries;      /* Total Number of Rules entries
                                      (Number of rules * Number of ACL
filters)
                                     */
  u_int32_t total_rule_delta_entries;/* Total Number of Rules entries
                                      (Number of rules * Number of ACL
filters)
                                      */
  u_int32_t num_rule_entries;        /* Total number of rule entries being
sent
                                       to HSL in one call */
  u_int32_t num_rule_delta_entries;  /* Total number of rule delta entries
                                        being sent to HSL in one call */
#define HAL_APBF_RULE_DELTA_ENTRIES_MORE  (1<<0)
#define HAL_APBF_RULE_ENTRIES_MORE        (1<<1)
  u_int8_t  flags;
#define HAL_APBF_GLOBAL_VR                (1<<0)
#define HAL_APBF_INTF_ALL                 (1<<1)
#define HAL_APBF_INTF_L3_ALL              (1<<2)
#define HAL_APBF_INTF_NOT_ALL             (1<<3)
#define HAL_APBF_INTF_NOT_L3_ALL          (1<<4)
  u_int8_t  apbf_flag;
  u_int32_t num_ifindices;           /* Total number of interfaces on which
                                        this rule should be applied. */
  u_int32_t *ifindices;              /* An Array of interface indices */

  struct list *hal_apbf_rule_list_delta; /* List of APBF rules for a rule
group,
                                            being newly added, deleted,
updates,
                                            of type "struct hal_apbf_rule" */
```

```
    struct list *hal_apbf_rule_list;       /* List of APBF rules for a rule
  group,
                                              the new rule set of type
                                              "struct hal_apbf_rule" */
```

hal_apbf_rule_failures

This structure is for the number of APBF rule failures.

Type	Definition
num_entries	Number of failed ACL identifiers.
failed_entries	List of hal_apbf_failed_attributes structs.

Definition

```
struct hal_apbf_rule_failures
{
  u_int32_t num_entries;          /* Number of failed ACL ids */
  struct list *failed_entries;    /* List of
                                     struct hal_apbf_failed_attributes     */
#define HAL_APBF_FAILED_ENTRIES_MORE(1<<0)
  u_int8_t flags;
};
```

hal_apbf_group_status

This structure is for the HAL message for group status.

Type	Definition
free_entry	Free entries.
free_meter	Free meters.
free_count	Free counters.

Definition

```
struct hal_apbf_group_status
{
  u_int32_t free_entry;
  u_int32_t free_meter;
  u_int32_t free_count;
};
```

hal_if_counters

This data structure helps manage all interface counter functions. It is defined in hal/hal_if.h.

Definition

```
struct hal_if_counters
  {
    ut_int64_t out_errors;
    ut_int64_t out_discards;
    ut_int64_t out_mc_pkts;
    ut_int64_t out_uc_pkts;
    ut_int64_t in_discards;
    ut_int64_t good_octets_rcv;
    ut_int64_t bad_octets_rcv;
    ut_int64_t mac_transmit_err;
    ut_int64_t good_pkts_rcv;
    ut_int64_t bad_pkts_rcv;
    ut_int64_t brdc_pkts_rcv;
    ut_int64_t mc_pkts_rcv;
    ut_int64_t pkts_64_octets;
    ut_int64_t pkts_65_127_octets;
    ut_int64_t pkts_128_255_octets;
    ut_int64_t pkts_256_511_octets;
    ut_int64_t pkts_512_1023_octets;
    ut_int64_t pkts_1024_max_octets;
    ut_int64_t good_octets_sent;
    ut_int64_t good_pkts_sent;
    ut_int64_t excessive_collisions;
    ut_int64_t mc_pkts_sent;
    ut_int64_t brdc_pkts_sent;
    ut_int64_t unrecog_mac_cntr_rcv;
    ut_int64_t fc_sent;
    ut_int64_t good_fc_rcv;
    ut_int64_t drop_events;
    ut_int64_t undersize_pkts;
    ut_int64_t fragments_pkts;
    ut_int64_t oversize_pkts;
    ut_int64_t jabber_pkts;
    ut_int64_t mac_rcv_error;
    ut_int64_t bad_crc;
    ut_int64_t collisions;
    ut_int64_t late_collisions;
    ut_int64_t bad_fc_rcv;
    ut_int64_t port_in_overflow_frames;
    ut_int64_t port_out_overflow_frames;
    ut_int64_t port_in_overflow_discards;
    ut_int64_t in_filtered;
    ut_int64_t out_filtered;
    ut_int64_t mtu_exceed;
  };
```

hal_port_map

This data structure helps manage the port bit map function. It is defined in hal/hal_if.h.

Type	Definition
bitmap[HAL_BIT_MAP_MAX]	

Definition

```
    struct hal_port_map
    {
      u_int32_t bitmap[HAL_BIT_MAP_MAX];
};
```

hal_in4_addr

This data structure helps manage all IPv4 address functions. It is defined in hal/hal_types.h.

Definition

```
struct hal_in4_addr
{
  u_int32_t s_addr;
};
```

hal_in6_addr

This data structure helps manage all IPv6 address functions. It is defined in hal/hal_types.h.

Definition

```
struct hal_in6_addr
{
  union
  {
    u_int8_t  u6_addr8[16];
    u_int16_t u6_addr16[8];
    u_int32_t u6_addr32[4];
  } in6_u;
};
```

hal_msg_arp_update Struct

This data structure is defined in hal/hal_types.h.

Definition

```
struct hal_msg_arp_update
{
  struct hal_in4_addr ip_addr;
  unsigned char mac_addr[ETHER_ADDR_LEN];
  unsigned int  ifindex;
  u_int8_t is_proxy_arp;
  u_int8_t is_refresh;
  u_int32_t loopback_ifindex;
  u_int8_t is_notification;
};
```

hal_msg_ipv6_nbr_update

This data structure is defined in hal/hal_types.h.

Definition

```
struct hal_msg_ipv6_nbr_update
{
  struct hal_in6_addr addr;
  unsigned char mac_addr[ETHER_ADDR_LEN];
  unsigned int  ifindex;
  u_int8_t is_refresh;
```

```
      u_int32_t loopback_ifindex;
      u_int8_t is_notification;
    };
```

hal_msg_nh_ipv4_resolve

This structure is for the nexthop IPv4 resolved addresses.

Type	Definition
hal_in4_addr	IPv4 address.
num_nh_rules	Nexthop rules.
rule_ids	Rule identifier.

Definition

```
    struct hal_msg_nh_ipv4_resolve
    {
      unsigned short fib_id;
      char name[HAL_IFNAME_LEN + 1];
      struct hal_in4_addr addr;
      int num_nh_rules;
      u_int32_t *rule_ids;
    };
```

hal_msg_nh_ipv6_resolve

This structure is for the nexthop IPv6 resolved addresses.

Type	Definition
hal_in6_addr	IPv6 address.
num_nh_rules	Nexthop rules.
rule_ids	Rule identifier.

Definition

```
    struct hal_msg_nh_ipv6_resolve
    {
      unsigned short fib_id;
      char name[HAL_IFNAME_LEN + 1];
      struct hal_in6_addr addr6;
      int num_nh_rules;
      u_int32_t *rule_ids;
    };
```

if_ident

This data structure helps manage interface index functions. It is defined in lib/mpls_client/mpls_common.h.

Definition

```
    struct if_ident
    {
      u_int32_t if_index;
      char if_name[INTERFACE_NAMSIZ + 1];
```
};

hal_mpls_diffserv

This data structure helps manage MPLS diffserv functions. It is defined in hal/MPLS/hal_mpls_types.h.

Type	Definition
dscp_exp_map[8	DSCP-to-EXP mapping for ELSP
dscp	DSCP value for LLSP
af_set	AF set. Per Hop Behavior (PHB) scheduling class set.

Definition

```
    struct hal_mpls_diffserv
    {
      /* LSP type. */
      hal_mpls_lsp_type_t lsp_type;
      /* DSCP-to-EXP mapping for ELSP. */
      unsigned char dscp_exp_map[8];
      /* DSCP value for LLSP. */
      unsigned char dscp;
      /* AF set. Per Hop Behavior (PHB) scheduling class set. */
      unsigned char af_set;
    };
```

mpls_owner_fwd

This data structure helps manage MPLS owner functions. It is defined in hal/MPLS/hal_mpls_types.h.

Definition

```
    struct mpls_owner_fwd
    {
      /* IPI_PROTO_xxx */
      u_char protocol;
      union
      {
        struct rsvp_key_fwd r_key;
      } u;
    };
```

hal_fdb_entry

This data structure helps manage FDB (forwarding database) entry functions. It is defined in hal/L2/hal_l2.h.

Definition

```
    struct hal_fdb_entry
    {
      unsigned short vid;
      unsigned short svid;
      unsigned int ageing_timer_value;
      unsigned char mac_addr[ETHER_ADDR_LEN];
      int num;
      char is_local;
      unsigned char is_static;
      unsigned char is_forward;
      unsigned int port;
    };
```

hal_vlan_classifier_rule

This data structure helps manage VLAN classifier rules. It is defined in hal/L2/hal_l2.h.

Type	Definition
type	Type of classifier: protocol/MAC/subnet
short vlan_id	Destination vlan_id
rule_id	Rule identification number.
row_status	Row status for protocol group table.
union	Rule criteria.
mac[ETHER_ADDR_LEN]	Mac address
short ether_type	Protocol value
encaps	Packet layer 2 encapsulation
avl_tree *group_tree	Groups rule attached to

Definition

```
    struct hal_vlan_classifier_rule
    {
      int type;                     /* Type of classifier: Protocol/Mac/Subnet */
      unsigned short vlan_id;       /* Destination vlan_id                     */
      u_int32_t rule_id;            /* Rule identification number.             */
    #ifdef HAVE_SNMP
      u_int32_t row_status;         /* Row status for ProtocolGroupTable.      */
    #endif /* HAVE_SNMP */
      union                         /* Rule criteria.                          */
      {
       unsigned char mac[ETHER_ADDR_LEN];  /* Mac address.                     */
        struct
        {
          unsigned int addr;
          unsigned char masklen;
        }ipv4;
        struct
        {
         unsigned short ether_type;  /* Protocol value                         */
         unsigned int   encaps;      /* Packet L2 encapsulation.               */
        } protocol;
      } u;
      struct avl_tree *group_tree;     /* Groups rule attached to.  */
    };
```

hal_ipv6_nbr_cache_entry

This data structure helps manage IPv6 NBR cache entries. It is defined in hal/L3/hal_l3.h.

Definition

```
struct hal_ipv6_nbr_cache_entry
{
  struct hal_in6_addr addr;
  unsigned char mac_addr[ETHER_ADDR_LEN];
  unsigned int  ifindex;
  int static_flag;
};
```

hal_ipv4uc_nexthop

This data structure helps manage IPv4 unicast nexthop entries. It is defined in hal/L3/hal_l3.h.

Definition

```
struct hal_ipv4uc_nexthop
{
  unsigned int id;
  enum hal_ipuc_nexthop_type type;
  unsigned int egressIfindex;
  char *egressIfname;
  struct hal_in4_addr nexthopIP;
};
```

hal_port_info

This data structure helps manage port information. It is defined in hal/L2/hal_l2.h.

Definition

```
struct hal_port_info
{
  unsigned short      port_id;
  unsigned char       state;
};
```

General API

The following subsection list the general API for HAL.

Table 2-1: General API Functions

Functions	Description
hal_arp_cache_get	This function gets the ARP table.
hal_arp_del_all	This function gets the ARP table.

Table 2-1: General API Functions

Functions	Description
hal_arp_entry_refresh	This function deletes all dynamic and/or static ARP entries.
hal_arp_entry_del	This function deletes all dynamic and/or static ARP entries.
hal_arp_entry_ageout_set	This function deletes all dynamic and/or static ARP entries.
hal_deinit	This function deinitializes the HAL component.
hal_init	This function initializes the HAL component.
hal_hw_reg_get	This function gets the value of the hardware register passed as input.
hal_hw_reg_set	This function sets the value of the hardware register passed as input.
hal_statistics_vlan_get	This function gets the statistics corresponding to the VLAN ID given as input.
hal_statistics_port_get	This function gets the statistics corresponding to the port ID given as input.
hal_statistics_host_get	This function gets the statistics corresponding to the host ID given as input.
hal_statistics_clear	This function clears the statistics.

hal_arp_cache_get

This function gets the ARP table starting at the next address of the IP address and the number of entries. It returns the number of entries found. The memory must be allocated by the caller before calling this API.

Syntax

```
int
hal_arp_cache_get(unsigned short fib_id, struct pal_in4_addr *ipaddr,
                  int count, struct hal_arp_cache_entry *cache)
```

Input Parameters

fib_id	FIB table ID
ipaddr	IP address
count	Request count
cache	ARP cache

Output Parameters

None

Return Value

Number of entries

Zero (0) for number entries

hal_arp_del_all

This function deletes all dynamic and/or static ARP entries

Syntax

```
int
hal_arp_del_all (unsigned short fib_id, u_char clr_flag)
```

Input Parameters

fib_id	FIB Table ID
clr_fl	Flag to indicate dynamic or static ARP entries

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds.

hal_arp_entry_refresh

This function refreshes ARP entries.

Syntax

```
int
hal_arp_entry_refresh (struct pal_in4_addr *ipaddr,
                       unsigned char *mac_addr,
                       u_int32_t ifindex)
```

Input Parameters

ipaddr	IP address.
mac_addr	MAC address.
ifindex	Interface index.

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds.

hal_arp_entry_del

This function deletes all ARP entries.

Syntax

```
int
hal_arp_entry_del(struct pal_in4_addr *ipaddr,
                  unsigned char *mac_addr, u_int32_t ifindex,
                  int is_notification)
```

Input Parameters

`ipaddr`	IP address.
`mac_addr`	MAC address.
`ifindex`	Interface index.
`is_notification`	

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds.

hal_arp_entry_ageout_set

This function sets the ARP entry age out value.

Syntax

```
int
hal_arp_entry_ageout_set (struct pal_in4_addr *ipaddr,
                          unsigned char *mac_addr,
                          u_int32_t ifindex)
```

Input Parameters

`ipaddr`	IP address.
`mac_addr`	MAC address.
`ifindex`	Interface index.

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds.

hal_deinit

This function deinitializes the HAL component.

Syntax

```
int
hal_deinit (struct lib_globals *zg);
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_init

This function initializes the HAL component.

Syntax

```
int
hal_init (struct lib_globals *zg);
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_hw_reg_get

This function gets the value of the hardware register passed as input.

Syntax

```
int
hal_hw_reg_get (u_int32_t reg_addr, struct hal_reg_addr *reg);
```

Input Parameters

`reg_addr`	Address of the hardware register

Output Parameters

`reg`	The value of the register is populated in this parameter.

Return Values

HAL_ERROR if unable to get the value of the register

HAL_SUCCESS

hal_hw_reg_set

This function sets the value of the hardware register passed as input.

Syntax

```
int
hal_hw_reg_set (u_int32_t reg_addr, u_int32_t value);
```

Input Parameters

`reg_addr`	Address of the hardware register
`value`	Value to be set on the register

Output Parameters

None

Return Values

HAL_ERROR if unable to set the value of the register

HAL_SUCCESS

hal_statistics_vlan_get

This function gets the statistics corresponding to the VLAN ID given as input.

Syntax

```
int
hal_statistics_vlan_get (u_int32_t vlan_id, struct hal_stats_vlan *vlan);
```

Input Parameters

`vlan_id`	VLAN ID for which statistics are to be obtained.

Output Parameters

`vlan`	The statistics will be populated in this output variable.

Return Values

HAL_ERROR if unable to get the statistics

HAL_SUCCESS

hal_statistics_port_get

This function gets the statistics corresponding to the port ID given as input.

Syntax

```
int
hal_statistics_port_get (u_int32_t port_id, struct hal_stats_port *port);
```

Input Parameters

`port_id`	Port ID for which statistics are to be obtained

Output Parameters

`port`	The statistics will be populated in this output variable.

Return Values

HAL_ERROR if unable to get the statistics

HAL_SUCCESS

hal_statistics_host_get

This function gets the host statistics.

Syntax

```
int
hal_statistics_host_get (struct hal_stats_host *host);
```

Input Parameters

None

Output Parameters

`host`	The statistics will be populated in this output variable.

Return Values

HAL_ERROR if unable to get the statistics

HAL_SUCCESS

hal_statistics_clear

Clear the statistics corresponding to the port ID given as input.

Syntax

```
int
hal_statistics_clear (u_int32_t port_id);
```

Input Parameters

`port_id`	Port ID for which statistics are to be obtained

Output Parameters

None

Return Values

HAL_ERROR if unable to clear the statistics

HAL_SUCCESS

hal_msg_decode_nh_ipv4_resolve

This function decodes IPv4 NH resolve messages.

Syntax

```
    int
    hal_msg_decode_nh_ipv4_resolve(u_char **pnt, u_int32_t *size,
                                   struct hal_msg_nh_ipv4_resolve *msg)
```

Input Parameters

`**pnt`	Message pointer.
`size`	Message size.
`msg`	Pointer for storing the message.

Output Parameters

None

Returns

*pnt - sp;

hal_msg_decode_nh_ipv6_resolve

This function decodes IPv6 NH resolve messages.

Syntax

```
    int
    hal_msg_decode_nh_ipv6_resolve(u_char **pnt, u_int32_t *size,
                                   struct hal_msg_nh_ipv6_resolve *msg)
```

Input Parameters

`**pnt`	Message pointer.
`size`	Message size.
`msg`	Pointer for storing the message.

Output Parameters

None

Returns

*pnt - sp;

APBF API

The following subsection includes the APBF functions.

hal_apbf_group_create

This function creates a qualifier group with an input of “qsets.“

Syntax

```
    int
    hal_apbf_group_create (u_int32_t *grp_id, u_int16_t prio,
                           struct smi_apbf_filter_mask *qset, u_int8_t pipe_stage)
```

Input Parameters

`Prio` Priority of the group identifier.

`qset` Mask of qualifiers with which group shall be created.

`pipe_stage` Pipe stage.

Output Parameters

`group_id` Group identifier.

Returns

HSL_SUCCESS on success.

HSL_FAILURE on failure.

hal_apbf_group_delete

This function deletes qualifier group with an input group ID.

Syntax

```
int
hal_apbf_group_delete (u_int32_t grp_id, u_int8_t delete)
```

Input Parameters

`group_id` Group identifier.

`delete` Delete value, including:

- `TRUE` All field entries existing in the qualifier group are deleted. The qualifier group is also deleted in the hardware.
- `FALSE` If field entries do not exist in the group, the group is deleted. If entries exist in the group, SMI_APBF_ERROR_HW_FAILURE error is generated.

Output Parameters

None

Returns

HSL_SUCCESS on success.

HSL_FAILURE on failure.

hal_apbf_group_status_get

This function gets the status of the free hardware resources within a qualifier group.

Syntax

```
int
hal_apbf_group_status_get (u_int32_t grp_id,
                          struct hal_apbf_group_status *grp_status)
```

Input Parameters

`group_id` Group identifier.

Output Parameters

`grp_status`	Group status.

Returns

HSL_SUCCESS on success.

HSL_FAILURE on failure.

hal_apbf_rule_apply

This function creates or deletes any APBF rule related information in the hardware. It calls the _hal_apbf_rule_apply' API in order to create the message, encode the message with rule information, and send the message to HSL.

Syntax

```
int
hal_apbf_rule_apply (int command,
                     struct hal_msg_apbf_rule *hal_rule_msg,
                     struct hal_apbf_rule_failures *failed_entryp)
```

Input Parameters

`command`	Command to install or remove a rule.
`hal_rule_msg`	HAL rule message.

Output Parameters:

`failed_entryp`	List of failed entries.

Returns

HSL_SUCCESS on success.

HSL_FAILURE on failure.

HAL_MSG_APBF_RULE_ADD message is used for rule creation.

HAL_MSG_APBF_RULE_DELETE message is used for rule deletion.

hal_msg_encode_apbf_rule_apply

This function encodes the message before sending to HSL for installing/uninstalling APBF rules in the hardware. This function encodes only the configured information for the qualifiers and actions in the hal_apbf_rule API in the message. It will keep encoding rule related information until the packet buffer does not exceed the limit.

Syntax

```
int
hal_msg_encode_apbf_rule_apply (u_char **pnt, u_int32_t *size,
                                u_int32_t *num_entries,
                                struct hal_msg_apbf_rule *msg)
```

Input Parameters

`*msg`	Pointer to APBF rule message

Output Parameters:

`**pnt`	Pointer to packet.
`*num_entries`	Number of entries.
`size`	Pointer to length of the packet.

Returns:

HAL_SUCCESS on success.

HAL._ERROR on failure.

hal_apbf_rule_status_get

This function creates a message and sends it to HSL to get the APBF rule counter information from the hardware.

Syntax

```
int
hal_apbf_rule_status_get (struct hal_apbf_rule_status *rule_status,
                          struct hal_apbf_acl_id_status *status)
```

Input Parameters

`rule_status`	Rule status key.
`hal_rule_msg`	HAL rule message.

Output Parameters:

`Status`	64-bit count. Number of packets that hit a particular rule.

Returns

HSL_SUCCESS on success.

HSL_FAILURE on failure.

Bridge API

The following subsection includes the bridge API functions.

Table 2-2: Bridge API Functions

Functions	Description
hal_bridge_init	This function initializes the bridging hardware layer component.
hal_bridge_deinit	This function deinitializes the bridging hardware layer component.
hal_bridge_add	This function adds a bridge instance.
hal_bridge_delete	This function deletes a bridge instance.
hal_bridge_change_vlan_type	This function changes the type of the bridge.
hal_bridge_set_state	This function sets the state of the bridge.

Table 2-2: Bridge API Functions

Functions	Description
hal_bridge_set_ageing_time	This function sets the ageing time for a bridge.
hal_bridge_set_learning	This function sets the learning for a bridge.
hal_bridge_add_port	This function adds a port to a bridge.
hal_bridge_delete_port	This function deletes a port from a bridge.
hal_bridge_set_port_state	This function sets the port state of a bridge port.
hal_bridge_add_instance	This function adds an instance to a bridge.
hal_bridge_delete_instance	This function deletes an instance from a bridge.
hal_bridge_add_vlan_to_instance	This function adds a VLAN to an instance in a bridge.
hal_bridge_delete_vlan_from_instance	This function deletes a VLAN from an instance in a bridge.
hal_bridge_set_learn_fwd	This function sets the learn and forwarding flag for a port.
hal_bridge_set_proto_process_port	This function sets a protocol data unit to be tunnelled/discarded in a port.
hal_bridge_flush_fdb_by_port	This function flushes all forwarding database (FDB) entries for a port.
hal_bridge_flush_dynamic_fdb_by_mac	This function deletes all the dynamic entries with a given MAC address.

hal_bridge_init

This function initializes the bridging hardware layer component.

Syntax

```
int
hal_bridge_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INIT when function fails

HAL_SUCCESS when function succeeds

hal_bridge_deinit

This function deinitializes the bridging hardware layer component.

Syntax

```
int
hal_bridge_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_DEINIT

HAL_SUCCESS when function succeeds

hal_bridge_add

This function adds a bridge instance.

Syntax

```
int
hal_bridge_add (char *name, unsigned int is_vlan_aware, enum hal_bridge_type type,
                unsigned char edge, unsigned char beb, unsigned char *mac)
```

Input Parameters

`name`	Bridge name
`is_vlan_aware`	VLAN aware
`type`	Bridge type
`edge`	Edge bridge
`beb`	Backbone Edge bridge (Unused)
`mac`	MAC address (Unused)

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_EXISTS

HAL_ERR_BRIDGE_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_bridge_delete

This function deletes a bridge instance.

Syntax

```
int
```

```
hal_bridge_delete (char *name);
```

Input Parameters

`name`	Bridge name

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_NOT_EXISTS

HAL_ERR_BRIDGE_DELETE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_bridge_change_vlan_type

This function changes the type of the bridge.

Syntax

```
int
hal_bridge_change_vlan_type (char *name, int is_vlan_aware,
                             u_int8_t type);
```

Input Parameters

`name`	bridge name
`is_vlan_aware`	Bridge is VLAN aware
`type`	Type of bridge

Output Parameters

None

Return Value

HAL_SUCCESS when function succeeds

NEGATIVE VALUE IS RETURNED

hal_bridge_set_state

This function sets the state of the bridge. If the bridge is disabled it behaves like a dumb switch.

Syntax

```
int
hal_bridge_set_state (char *name, u_int16_t enable);
```

Input Parameters

`name`	Bridge name
`enable`	Enable/disable spanning tree

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_set_ageing_time

This function sets the ageing time for a bridge.

Syntax

```
int
hal_bridge_set_ageing_time (char *name, u_int32_t ageing_time);
```

Input Parameters

`name`	Bridge name
`ageing_time`	Ageing time in seconds

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_set_learning

This function sets the learning for a bridge.

Syntax

```
int
hal_bridge_set_learning (char *name, int learning);
```

Input Parameters

`name`	Bridge name
`learning`	Whether the bridge is a learning bridge or not

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_add_port

This function adds a port to a bridge.

Syntax

```
int
hal_bridge_add_port (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_PORT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_delete_port

This function deletes a port from a bridge.

Syntax

```
int
hal_bridge_delete_port (char *name, int ifindex);
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_set_port_state

This function sets the port state of a bridge port.

Syntax

```
int
hal_bridge_set_port_state (char *bridge_name,
                           int ifindex, int instance, int state)
```

Input Parameters

`bridge_name`	Bridge name
`ifindex`	Port interface index
`instance`	Instance number
`state`	Port state

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_add_instance

This function adds an instance to a bridge.

Syntax

```
int
hal_bridge_add_instance (char * name, int instance);
```

Input Parameters

`name`	Bridge name
`instance`	Instance number

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INSTANCE_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_delete_instance

This function deletes the instance from the bridge.

Syntax

```
int
hal_bridge_delete_instance (char * name, int instance);
```

Input Parameters

`name`	Bridge name
`instance`	Instance number

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INSTANCE_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_add_vlan_to_instance

This function adds a VLAN to an instance in a bridge.

Syntax

```
int
hal_bridge_add_vlan_to_instance (char * name, int instance, unsigned short vid);
```

Input Parameters

`name`	Bridge name
`instance`	Instance number
`vid`	VLAN ID

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INSTANCE_NOT_EXISTS

HAL_ERR_BRIDGE_VLAN_NOT_FOUND

HAL_SUCCESS when function succeeds

hal_bridge_delete_vlan_from_instance

This function deletes a VLAN from an instance in a bridge.

Syntax

```
int
hal_bridge_delete_vlan_from_instance (char *name, int instance, unsigned short vid)
```

Input Parameters

`name`	Bridge name
`instance`	Instance number
`vid`	VLAN ID

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INSTANCE_NOT_EXISTS

HAL_ERR_BRIDGE_VLAN_NOT_FOUND

HAL_SUCCESS when function succeeds

hal_bridge_set_learn_fwd

This function sets the learn and forwarding flag for a port.

Syntax

```
int
hal_bridge_set_learn_fwd (const char *const bridge_name,const int ifindex,
                          const int instance, const int learn, const int forward)
```

Input Parameters

`bridge_name`	Bridge name
`ifindex`	Port interface index
`instance`	Instance number
`learn`	Enable learning
`forward`	Enable forwarding

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_INSTANCE_NOT_EXISTS

HAL_ERR_BRIDGE_VLAN_NOT_FOUND

HAL_SUCCESS when function succeeds

hal_bridge_set_proto_process_port

This function configures a particular protocol data unit to be tunnelled or discarded in a customer facing port.

Syntax

```
int
hal_bridge_set_proto_process_port (const char *const bridge_name,
                                   const int ifindex,
                                   enum hal_l2_proto proto,
                                   enum hal_l2_proto_process process,
                                   u_int16_t vid);
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index
`proto`	Protocols whose PDUs have to be discarded/tunnelled

process	Whether PDUs have to be discarded/tunnelled
vid	LAN ID

Output Parameters

None

Return Value

HAL_ERR_BRIDGE_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_bridge_flush_fdb_by_port

This function flushes all forwarding database (FDB) entries for a port.

Syntax

```
int
hal_bridge_flush_fdb_by_port(char *name, unsigned int ifindex, unsigned int instance,
                             unsigned short vid, unsigned short svid)
```

Input Parameters

name	Bridge name
ifindex	Port interface index
instance	
vid	VLAN ID
svid	S-VLAN ID

Output Parameters

None

Return Value

HAL_ERR_L2_FDB_NO_ENTRY

HAL_ERR_L2_FDB_ENTRY

HAL_SUCCESS when function succeeds

hal_bridge_flush_dynamic_fdb_by_mac

This function deletes all the dynamic entries with a given MAC address.

Syntax

```
int
hal_bridge_flush_dynamic_fdb_by_mac (char *bridge_name,
                                     const unsigned char * const mac,
                                     int maclen)
```

Input Parameters

bridge_name	Bridge name
mac	MAC address of the dynamic entry to be deleted.
maclen	MAC address length

Output Parameters

None

Return Value

HAL_SUCCESS when function succeeds

Flow Control API

The following subsection includes the flow control API functions.

Table 2-3: Flow Control API Functions

Functions	Description
hal_flow_control_init	This function initializes the flow control hardware layer.
hal_flow_control_deinit	This function deinitializes the flow control hardware layer.
hal_flow_control_set	This function sets the flow control properties of a port.
hal_flow_ctrl_pause_watermark_set	This function sets the watermark pause flow control property of a port.
hal_flow_control_statistics	This function gets the flow control statistics for a port.

hal_flow_control_init

This function initializes the flow control hardware layer.

Syntax

```
int
hal_flow_control_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_FLOWCTRL_INIT when function fails

HAL_SUCCESS when function succeeds

hal_flow_control_deinit

This function deinitializes the flow control hardware layer.

Syntax

```
int
hal_flow_control_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_FLOWCTRL_DEINIT when function fails

HAL_SUCCESS when function succeeds

hal_flow_control_set

This function sets the flow control properties of a port.

Syntax

```
int
hal_flow_control_set (unsigned int ifindex, unsigned char direction);
```

Input Parameters

`ifindex`	Port interface index
`direction`	HAL_FLOW_CONTROL_(OFF|SEND|RECEIVE)

Output Parameters

None

Return Value

HAL_ERR_FLOW_CONTROL_SET when function fails

HAL_SUCCESS when function succeeds

hal_flow_ctrl_pause_watermark_set

This function sets the watermark pause flow control property of a port.

Syntax

```
int
hal_flow_ctrl_pause_watermark_set (u_int32_t port, u_int16_t wm_pause);
```

Input Parameters

`ifindex`	Port interface index

`wm_pause`	WaterMark Pause

Output Parameters

None

Return Value

HAL_ERR_FLOW_CONTROL_SET when function fails

HAL_SUCCESS when function succeeds

hal_flow_control_statistics

This function gets the flow control statistics for a port.

Syntax

```
int
hal_flow_control_statistics (unsigned int ifindex, unsigned char *direction,
                             int *rxpause, int *txpause)
```

Input Parameters

`ifindex`	Port interface index

Output Parameters

`direction`	HAL_FLOW_CONTROL_(OFF|SEND|RECEIVE)
`rxpause`	Number of received pause frames
`txpause`	Number of transmitted pause frames

Return Value

HAL_ERR_FLOW_CONTROL when function fails

HAL_SUCCESS when function succeeds

GARP API

The following subsection includes the (GARP) Generic Attribute Registration Protocol functions.

Table 2-4: GARP API Functions

Functions	Description
hal_garp_set_bridge_type	This function sets the bridge to GMRP/GVRP enabled or disabled.

hal_garp_set_bridge_type

This function sets the bridge to GMRP/GVRP enabled or disabled.

Syntax

```
void
hal_garp_set_bridge_type (char *bridge_name, unsigned long garp_type, int enable)
```

Input Parameters

`bridge_name`	The name of the bridge
`garp_type`	gvrp = 02
	gmrp = 01
`enable`	True or false

Output Parameters

None

Return Value

Void

Forwarding Information Base API

The following subsection includes the forwarding information base (FIB) functions for layer 3 protocols.

Table 2-5: FIB API Functions

Functions	Description
hal_fib_create	This function creates a FIB in the forwarding plane.
hal_fib_delete	This function deletes a FIB in the forwarding plane.

hal_fib_create

This function creates a FIB in the forwarding plane for the provided FIB ID.

Syntax

```
int
hal_fib_create (unsigned int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_FIB_EXISTS

HAL_SUCCESS when function succeeds

< 0 when function fails

hal_fib_delete

This function deletes a FIB in the forwarding plane for the provided FIB ID.

Syntax

```
int
hal_fib_delete (unsigned int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_FIB_NOT_EXISTS

HAL_SUCCESS when function succeeds

< 0 when function fails

IGMP Snooping API

The following subsection includes the IGMP (Internet Group Management Protocol) functions.

Table 2-6: IGMP API Functions

Functions	Description
hal_igmp_snooping_if_enable	This function enables IGMP snooping for the interface
hal_igmp_snooping_if_disable	This function disables IGMP snooping for the interface
hal_igmp_snooping_init	This function initializes the reception of IGMP packets for IGMP snooping.
hal_igmp_snooping_deinit	This function deinitializes the reception of IGMP packets for IGMP snooping.
hal_igmp_snooping_enable	This function enables IGMP snooping for the bridge.
hal_igmp_snooping_disable	This function disables IGMP snooping for the bridge.
hal_igmp_snooping_add_entry	This function adds a multicast entry for a given VLAN.
hal_igmp_snooping_delete_entry	This function deletes a multicast entry from a given VLAN.

hal_igmp_snooping_if_enable

This function enables IGMP snooping for the interface

Syntax

```
int
hal_igmp_snooping_if_enable (char *name, unsigned int ifindex)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_ENABLE when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_if_disable

This function disables IGMP snooping for the interface.

Syntax

```
int
hal_igmp_snooping_if_disable (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_DISABLE when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_init

This function initializes the reception of IGMP packets for IGMP snooping.

Syntax

```
int
hal_igmp_snooping_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_INIT when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_deinit

This function deinitializes the reception of IGMP packets for IGMP snooping.

Syntax

```
int
hal_igmp_snooping_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_INIT when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_enable

This function enables IGMP snooping for the bridge.

Syntax

```
int
hal_igmp_snooping_enable (char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_ENABLE when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_disable

This function disables IGMP snooping for the bridge.

Syntax

```
int
hal_igmp_snooping_disable (char *bridge_name)
```

Input Parameters

`name`	Bridge name

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_DISABLE when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_add_entry

This function adds a multicast entry (source, group) for a given VLAN. If the group doesn't exist, a new one is created. If the group exists, the list of ports is added to the entry.

Syntax

```
int
hal_igmp_snooping_add_entry (char *bridge_name,
                             struct hal_in4_addr *src,
                             struct hal_in4_addr *group,
                             char is_exclude,
                             int vid,
                             int svid,
                             int count,
                             u_int32_t *ifindexes)
```

Input Parameters

`bridge_name`	Bridge name
`src`	Multicast source address
`group`	Multicast group address
`is_exclude`	
`vid`	VLAN ID
`svid`	Service VLAN ID
`count`	Count of ports to add
`ifindexes`	Array of ports to add

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_ENTRY_ERR when function fails

HAL_SUCCESS when function succeeds

hal_igmp_snooping_delete_entry

This function deletes a multicast entry (source, group) for a given VLAN. If the group doesn't exist, a error is returned. If the group exists, the list of ports are deleted from the multicast entry. If it is the last port for the multicast entry, the multicast entry is deleted as well.

Syntax

```
int
hal_igmp_snooping_delete_entry (char *bridge_name,
                                struct hal_in4_addr *src,
                                struct hal_in4_addr *group,
                                char is_exclude,
                                int vid,
                                int svid,
                                          int count,
                                          u_int32_t *ifindexes)
```

Input Parameters

`bridge_name`	Bridge name
`src`	Multicast source address
`group`	Multicast group address
`is_exclude`	
`vid`	VLAN ID
`svid`	Service VLAN ID
`count`	Count of ports to delete
`ifindexes`	Array of ports to delete

Output Parameters

None

Return Value

HAL_ERR_IGMP_SNOOPING_ENTRY_ERR when function fails

HAL_SUCCESS when function succeeds

Interface API

The following subsection describes the interface functions.

Table 2-7: Interface API Functions

Functions	Description
hal_if_bind_fib	This function binds an interface to a FIB fib_id in the forwarding plane.
hal_if_unbind_fib	This function unbinds an interface to a FIB fib_id in the forwarding plane.
hal_if_get_list	This function gets the list of interfaces from the interface manager.
hal_if_get_metric	This function gets the metric for an interface.
hal_if_get_mtu	This function gets the Maximum Transmission Unit (MTU) for an interface.
hal_if_set_mtu	This function sets the Maximum Transmission Unit (MTU) for an interface.
hal_if_set_arp_ageing_timeout	This function sets the ARP ageing timeout for an interface.
hal_if_get_arp_ageing_timeout	This function gets the ARP ageing timeout for an interface.
hal_if_get_duplex	This function gets the duplex mode for the interface.
hal_if_set_duplex	This function sets the duplex mode for the interface.
hal_if_set_autonego	This function sets the mode to auto-negotiate for an interface.

Table 2-7: Interface API Functions

Functions	Description
hal_if_get_hwaddr	This function gets the hardware address for an interface.
hal_if_set_hwaddr	This function sets the hardware address for an interface.
hal_if_sec_hwaddrs_set	This function sets the list of secondary MAC addresses for an interface.
hal_if_sec_hwaddrs_add	This function adds the secondary hardware addresses to the list of a MAC.
hal_if_sec_hwaddrs_delete	This function deletes the secondary hardware addresses from the list of a MAC.
hal_if_flags_get	This function gets the flags for an interface.
hal_if_flags_set	This function sets the flags for an interface.
hal_if_flags_unset	This function unsets the flags from an interface.
hal_if_get_bw	This function gets the bandwidth for the interface.
hal_if_set_bw	This function sets the bandwidth for the interface.
hal_if_delete_done	This function deletes interface from the interface manager.
hal_if_set_port_type	This function sets the port type for an interface.
hal_if_svi_create	This function creates a Switch Virtual Interface (SVI) for a specific VLAN.
hal_if_svi_delete	This function deletes an SVI from a specific VLAN.
hal_if_get_counters	This function gets the interface statistics for a specific interface index.
hal_if_set_mdix	This function sets MDIX crossover for specified interface index.
hal_if_set_portbased_vlan	This function sets members to a port-based VLAN.
hal_if_set_port_egress	This function sets the port egress type.
hal_if_set_force_vlan	This function sets the port VLAN.
hal_if_set_ether_type	This function sets the port Ethernet type.
hal_if_set_sw_reset	This function resets the HSL driver.
hal_if_set_learn_disable	This function sets learning to disable on the interface.
hal_if_get_learn_disable	This function gets the learn disable status for the interface.
hal_if_ipv4_address_add	This function adds an IPv4 address to a layer 3 interface.
hal_if_ipv4_address_delete	This function deletes an IPv4 address from a layer 3 interface.
hal_if_ipv6_address_add	This function adds an IPv6 address to a layer 3 interface.
hal_if_ipv6_address_delete	This function deletes an IPv6 address from a layer 3 interface.

hal_if_bind_fib

This function is called to bind a interface to a FIB fib_id in the forwarding plane.

Syntax

```
int
hal_if_bind_fib (u_int32_t ifindex, u_int32_t fib);
```

Input Parameters

`ifindex`	Interface index
`fib`	FIB ID

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_unbind_fib

This function is called to unbind a interface to a FIB fib_id in the forwarding plane.

Syntax

```
int
hal_if_unbind_fib (u_int32_t ifindex, u_int32_t fib);
```

Input Parameters

`ifindex`	Interface index
`fib`	FIB ID

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_list

This function gets the list of interfaces from the interface manager.

Syntax

```
int
hal_if_get_list (void);
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_metric

This function gets the metric for an interface.

Syntax

```
int
hal_if_get_metric (char *ifname, unsigned int ifindex, int *metric);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`metric`

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_mtu

This function gets the Maximum Transmission Unit (MTU) for an interface.

Syntax

```
int
hal_if_get_mtu (char *ifname, unsigned int ifindex, int *mtu)
```

Input Parameters

`ifname`	Interface name

Output Parameters

`mtu`	Maximum transmission unit

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_mtu

This function sets the MTU for an interface.

Syntax

```
int
hal_if_set_mtu (char *ifname, unsigned int ifindex, int mtu);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`mtu`	Maximum transmission unit

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_arp_ageing_timeout

This function sets the Address Resolution Protocol (ARP) ageing timeout for an interface.

Syntax

```
int
hal_if_set_arp_ageing_timeout (char *ifname, unsigned int ifindex, int
                               arp_ageing_timeout)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`arp_ageing_timeout`	ARP ageing time-out value

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_arp_ageing_timeout

This function gets the ARP ageing timeout for an interface.

Syntax

```
int
hal_if_get_arp_ageing_timeout (char *ifname, unsigned int ifindex, int
                              *arp_ageing_timeout)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`arp_ageing_timeout`

ARP aging time-out value

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_duplex

This function gets the duplex mode for the interface.

Syntax

```
int
hal_if_get_duplex (char *ifname, unsigned int ifindex, int *duplex)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`duplex`	Duplex mode

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_duplex

This function sets the duplex mode for an interface.

Syntax

```
int
hal_if_set_duplex (char *ifname, unsigned int ifindex, int duplex);
```

Input Parameters

`ifname`	Interface name

`ifindex`	Interface index
`duplex`	Duplex mode

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_autonego

This function sets the mode to auto-negotiate for an interface.

Syntax

```
int
hal_if_set_autonego (char *ifname, unsigned int ifindex, int autonego);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`autonego`	Auto-negotiate

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_hwaddr

This function gets the hardware address for an interface. This is the Media Access Control (MAC) address, in case of Ethernet requirements. The caller has to provide a buffer large enough to hold the address.

Syntax

```
int
hal_if_get_hwaddr (char *ifname, unsigned int ifindex,
                   u_char *hwaddr, int *hwaddr_len)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`hwaddr`	Hardware address

`hwaddr_len`	Hardware address length

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_hwaddr

This function sets the hardware address for an interface. This is the MAC address in case of Ethernet requirements.

Syntax

```
int
hal_if_set_hwaddr (char *ifname, unsigned int ifindex,
                   u_int8_t *hwaddr, int hwlen)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`hwaddr`	Hardware address
`hwlen`	Hardware address length

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_sec_hwaddrs_set

This function sets the list of secondary MAC addresses for an interface.

Syntax

```
int
hal_if_sec_hwaddrs_set (char *ifname, unsigned int ifindex,
                        int hw_addr_len, int nAddrs, unsigned char **addresses)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`hw_addr_len`	Hardware address length
`nAddrs`	Number of MAC addresses
`addresses`	Array of MAC addresses

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_sec_hwaddrs_add

This function adds the secondary hardware addresses to the list of MAC addresses for an interface.

Syntax

```
int
hal_if_sec_hwaddrs_add (char *ifname, unsigned int ifindex,
                        int hw_addr_len, int nAddrs, unsigned char **addresses)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`hw_addr_len`	Hardware address length
`nAddrs`	Number of MAC addresses
`addresses`	Array of MAC addresses

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_sec_hwaddrs_delete

This function deletes the secondary hardware addresses from the list of received MAC addresses for an interface.

Syntax

```
int
hal_if_sec_hwaddrs_delete (char *ifname, unsigned int ifindex,
                           int hw_addr_len, int nAddrs, unsigned char **addresses)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`hw_addr_len`	Hardware address length
`nAddrs`	Number of MAC addresses
`addresses`	Array of MAC addresses

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_flags_get

This function gets the flags for an interface.

Syntax

```
int
hal_if_flags_get (char *ifname, unsigned int ifindex, u_int32_t *flags)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`flags`	One of the following: IFF_RUNNING IFF_UP IFF_BROADCAST IFF_LOOPBACK

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_flags_set

This function sets the flags for an interface.

Syntax

```
int
hal_if_flags_set (char *ifname, unsigned int ifindex, unsigned int flags);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`flags`	Flags to set to one of the following: IFF_RUNNING IFF_UP

	IFF_BROADCAST
	IFF_LOOPBACK

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_flags_unset

This function unsets the flags for an interface.

Syntax

```
int
hal_if_flags_unset (char *ifname, unsigned int ifindex, unsigned int flags);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`flags`	Unset flags; one of the following:
	IFF_RUNNING
	IFF_UP
	IFF_BROADCAST
	IFF_LOOPBACK

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_bw

This function gets the bandwidth for the interface.

Syntax

```
int
hal_if_get_bw (char *ifname, unsigned int ifindex, u_int32_t *bandwidth)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

`bandwidth`	Interface bandwidth in bytes per second

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_bw

This function sets the bandwidth for an interface.

Syntax

```
int
hal_if_set_bw (char *ifname, unsigned int ifindex, unsigned int bandwidth)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`bandwidth`	Interface bandwidth

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_delete_done

This function deletes interface from the interface manager after getting acknowledgement from the protocol modules.

Syntax

```
int
hal_if_delete_done(char *ifname, u_int16_t ifindex);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_port_type

This function sets the port type for an interface.

Syntax

```
int
hal_if_set_port_type (char *name, unsigned int ifindex,
                       enum hal_if_port_type type, unsigned int *retifindex)
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index
`type`	Port type, either: ROUTER SWITCH

Output Parameters

`retifindex`	Interface index of the new type of interface

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_svi_create

This function creates a Switch Virtual Interface (SVI) for a specific VLAN. The VLAN information is embedded in the name of the interface.

Syntax

```
int
hal_if_svi_create (char *name, unsigned int *retifindex)
```

Input Parameters

`name`	Interface name

Output Parameters

`retifindex`	Interface index of the new type of interface

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_svi_delete

This function deletes the SVI for a specific VLAN.

Syntax

```
int
hal_if_svi_delete (char *name, unsigned int ifindex)
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_counters

This function gets the interface statistics for a specific interface index.

Syntax

```
int
hal_if_get_counters(unsigned int ifindex, struct hal_if_counters *if_stats)
```

Input Parameters

`ifindex`	Interface index.

Output Parameters

`if_stats`	The array of counters for interface.

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_mdix

This function sets MDIX crossover for specified interface index.

Syntax

```
int
hal_if_set_mdix(unsigned int ifindex, unsigned int mdix);
```

Input Parameters

`ifindex`	Interface index
`mdix`	MDIX crossover for an interface

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_portbased_vlan

This function sets members to a port-based VLAN.

Syntax

```
int
hal_if_set_portbased_vlan (unsigned int ifindex, struct hal_port_map pbitmap)
```

Input Parameters

`ifindex`	Interface index
`pbitmap`	Bit map for port-based VLAN members

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_port_egress

This function sets the port egress type.

Syntax

```
int
hal_if_set_port_egress (unsigned int ifindex, int egress);
```

Input Parameters

`ifindex`	Interface index
`egress`	Port egress mode

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_force_vlan

This function sets the port VLAN.

Syntax

```
int
hal_if_set_force_vlan (unsigned int ifindex, int vid);
```

Input Parameters

`ifindex`	Interface index
`vid`	Port VLAN ID

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_ether_type

This function sets the port Ethernet type.

Syntax

```
int
hal_if_set_ether_type (unsigned int ifindex, u_int16_t etype)
```

Input Parameters

`ifindex`	Interface index
`etype`	Ethernet type

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_sw_reset

This function resets the HSL driver.

Syntax

```
int
hal_if_set_sw_reset ()
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_set_learn_disable

This function sets learning to disable on the interface.

Syntax

```
int
hal_if_set_learn_disable (unsigned int ifindex, int enable);
```

Input Parameters

`ifindex`	Interface index
`enable`	Enable/disable

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_get_learn_disable

This function gets the learn disable status for the interface.

Syntax

```
int
hal_if_get_learn_disable (unsigned int ifindex, int* enable);
```

Input Parameters

`ifindex`	Interface index
`enable`	Enable/disable

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_ipv4_address_add

This function adds an IPv4 address to a layer 3 interface.

Syntax

```
int
hal_if_ipv4_address_add (char *ifname, unsigned int ifindex,
                         struct pal_in4_addr *ipaddr, unsigned char ipmask)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`ipaddr`	Interface IP address
`ipmask`	Mask length

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_ipv4_address_delete

This function deletes an IPv4 address from a layer 3 interface.

Syntax

```
int
hal_if_ipv4_address_delete (char *ifname, unsigned int ifindex,
                            struct pal_in4_addr *ipaddr, unsigned char ipmask)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`ipaddr`	Interface IP address
`ipmask`	Mask length

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_ipv6_address_add

This function adds an IPv6 address to a layer 3 interface.

Syntax

```
int
hal_if_ipv6_address_add (char *ifname, unsigned int ifindex,
                         struct pal_in6_addr *ipaddr, int ipmask,
                         unsigned char flags);
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`ipaddr`	Interface IP address
`ipmask`	Mask length
`flags`	Flags for the address

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_if_ipv6_address_delete

This function deletes the IPv6 address from a layer 3 interface.

Syntax

```
int
hal_if_ipv6_address_delete (char *ifname, unsigned int ifindex,
                            struct pal_in6_addr *ipaddr,
                            int ipmask)
```

Input Parameters

`ifname`	Interface name
`ifindex`	Interface index
`ipaddr`	Interface IP address
`ipmask`	Mask length

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

IPv4 API

The following subsection describes the IPv4 functions.

Table 2-8: IPV4 API Functions

Functions	Description
hal_ipv4_init	This function initializes the IPv4 hardware layer component.
hal_ipv4_deinit	This function deinitializes the IPv4 hardware layer component.
hal_ipv4_arp_add	This function adds an IPv4 ARP entry.
hal_ipv4_arp_del	This function deletes an IPv4 ARP entry.
hal_ipv4_uc_init	This function initializes the IPv4 unicast table for the specified FIB ID.
hal_ipv4_uc_deinit	This function deinitializes the IPv4 unicast table for the specified FIB ID.
hal_ipv4_uc_route_add	This function adds an IPv4 unicast route to the forwarding plane.
hal_ipv4_uc_route_delete	This function deletes an IPv4 unicast route from the forwarding plane.
hal_ipv4_uc_route_update	This function updates an IPv4 unicast route for the forwarding plane.
hal_ipv4_mc_init	This function initializes the IPv4 multicast table for the specified FIB ID.
hal_ipv4_mc_deinit	This function deinitializes the IPv4 multicast table for the specified FIB ID.
hal_ipv4_mc_pim_init	This function initializes PIM routing for the specified FIB ID.
hal_ipv4_mc_pim_deinit	This function deinitializes PIM routing for the specified FIB ID.
hal_ipv4_mc_get_max_vifs	This function gets the maximum number of VIFs supported.
hal_ipv4_mc_vif_add	This function creates a virtual interface (VIF).
hal_ipv4_mc_vif_delete	This function deletes a VIF.
hal_ipv4_mc_vif_addr_add	This function adds an address to a configured VIF.
hal_ipv4_mc_vif_addr_delete	This function deletes an address to a configured VIF.
hal_ipv4_mc_vif_set_physical_if	This function sets the physical interface index to a configured VIF.
hal_ipv4_mc_vif_set_flags	This function sets the VIF flags of a configured VIF.
hal_ipv4_mc_set_min_ttl_threshold	This function sets the minimum TTL a multicast packet must have to be forwarded.
hal_ipv4_mc_get_min_ttl_threshold	This function gets the minimum TTL a multicast packet is to be forwarded on a VIF.
hal_ipv4_mc_set_max_rate_limit	This function sets the maximum multicast bandwidth rate allowed on a VIF.

Table 2-8: IPV4 API Functions

Functions	Description
hal_ipv4_mc_get_max_rate_limit	This function gets the maximum multicast bandwidth rate allowed on this VIF.
hal_ipv4_mc_add_mfc	This function installs or modifies a multicast forwarding cache (MFC).
hal_ipv4_mc_delete_mfc	This function deletes a multicast forwarding cache (MFC).
hal_ipv4_mc_get_sg_count	This function gets the various counters per (S, G) entry.

hal_ipv4_init

This function initializes the IPv4 hardware layer component.

Syntax

```
int
hal_ipv4_init (void)
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_deinit

This function deinitializes the IPv4 hardware layer component.

Syntax

```
int
hal_ipv4_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_arp_add

This function adds an IPv4 ARP entry.

Syntax

```
int
hal_arp_entry_add(struct pal_in4_addr *ipaddr,
                  unsigned char *mac_addr,
                  u_int32_t ifindex,
                  u_int8_t is_proxy_arp)
```

Input Parameters

`ipaddr`	IP address
`mac_addr`	MAC address
`ifindex`	Interface index
`is_proxy_arp`	Is this proxy ARP

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_arp_del

This function deletes an IPv4 ARP entry.

Syntax

```
int
hal_arp_entry_del(struct pal_in4_addr *ipaddr,
                  unsigned char *mac_addr,
                  u_int32_t ifindex)
```

Input Parameters

`ipaddr`	IP address
`mac_addr`	MAC address
`ifindex`	Interface index

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_uc_init

This function initializes the IPv4 unicast table for the specified FIB ID.

Syntax

```
int
hal_ipv4_uc_init (unsigned short fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_uc_deinit

This function deinitializes the IPv4 unicast table for the specified FIB ID

Syntax

```
int
hal_ipv4_uc_deinit (unsigned short fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_uc_route_add

This function adds an IPv4 unicast route to the forwarding plane.

Syntax

```
int
```

```
hal_ipv4_uc_route_add (unsigned short fib,
                       struct pal_in4_addr *ipaddr,
                       unsigned char masklen,
                       unsigned short num, struct hal_ipv4uc_nexthop *nexthops)
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`masklen`	IP mask length
`num`	Number of nexthops
`nexthops`	List of nexthops

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_uc_route_delete

This function deletes an IPv4 route from the forwarding plane.

Syntax

```
int
hal_ipv4_uc_route_delete (unsigned short fib,
                         struct pal_in4_addr *ipaddr,
                         unsigned char masklen,
                         unsigned short num,
                         struct hal_ipv4uc_nexthop *nexthops)
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`masklen`	IP mask length
`num`	Number of nexthops
`nexthops`	List of nexthops

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_uc_route_update

This function updates an existing IPv4 route with the new nexthop parameters.

Syntax

```
int
hal_ipv4_uc_route_update (unsigned short fib,
                         struct pal_in4_addr *ipaddr,
                         unsigned char ipmasklen,
                         unsigned short numfib, struct hal_ipv4uc_nexthop *nexthopsfib,
                         unsigned short numnew, struct hal_ipv4uc_nexthop *nexthopsnew)
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`ipmasklen`	IP mask length
`numfib`	Number of nexthops in the FIB
`nexthopsfib`	List of nexthops in the FIB
`numnew`	Number of new or updated nexthops
`nexthopsnew`	List of new or updated nexthops in the FIB

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_init

This function initializes the IPv4 multicast table for the specified FIB ID.

Syntax

```
int
hal_ipv4_mc_init (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_deinit

This function deinitializes the IPv4 multicast table for the specified FIB ID.

Syntax

```
int
hal_ipv4_mc_deinit (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_pim_init

This function initializes PIM routing for the specified FIB ID.

Syntax

```
int
hal_ipv4_mc_pim_init (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_pim_deinit

This function stops PIM routing for the specified FIB ID.

Syntax

```
int
```

```
hal_ipv4_mc_pim_deinit (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_get_max_vifs

This function gets the maximum number of VIFs supported.

Syntax

```
int
hal_ipv4_mc_get_max_vifs (int *vifs);
```

Input Parameters

`vifs`	Max VIFs

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_add

This function creates a virtual interface (VIF).

Syntax

```
int
hal_ipv4_mc_vif_add (u_int32_t index, u_int32_t phy_ifindex,
    struct hal_in4_addr *loc_addr, struct hal_in4_addr *rmt_addr,
    u_int16_t flags)
```

Input Parameters

`index`	VIF index
`phy_ifindex`	Interface index
`loc_addr`	VIF local address

`rmt_addr`	VIF remote address (tunnel)
`flags`	VIF flags

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_EXISTS

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_delete

This function deletes a VIF.

Syntax

```
int
hal_ipv4_mc_vif_delete (u_int32_t index)
```

Input Parameters

`index`	VIF index

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_IPV4_MC_VIF_MAX_EXCEEDED

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_addr_add

This function adds an address to a configured VIF.

Syntax

```
int
hal_ipv4_mc_vif_addr_add (unsigned int index,
                          struct hal_in4_addr *addr,
                          struct hal_in4_addr *subnet,
                          struct hal_in4_addr *broadcast,
                          struct hal_in4_addr *peer)
```

Input Parameters

`index`	VIF index
`addr`	Address to add

`subnet`	Subnet address to add
`broadcast`	Broadcast address to add
`peer`	Peer address to add

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_addr_delete

This function deletes an address from a configured VIF.

Syntax

```
int
hal_ipv4_mc_vif_addr_delete (unsigned int index, struct hal_in4_addr *addr);
```

Input Parameters

`index`	VIF index
`addr`	Address to delete

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_IPV4_MC_VIF_ADDRESS_NOTFOUND

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_set_physical_if

This function sets the physical interface index to a configured VIF.

Syntax

```
int
hal_ipv4_mc_vif_set_physical_if (unsigned int index, unsigned int ifindex);
```

Input Parameters

`index`	VIF index
`ifindex`	Physical interface index

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_IPV4_MC_IF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv4_mc_vif_set_flags

This function sets the VIF flags of a configured VIF.

Syntax

```
int
hal_ipv4_mc_vif_set_flags (unsigned int ifindex,
                            unsigned char is_pim_register,
                            unsigned char is_p2p,
                            unsigned char is_loopback,
                            unsigned char is_multicast,
                            unsigned char is_broadcast);
```

Input Parameters

`index`	VIF index
`is_pim_register`	True if the VIF is a PIM register interface
`is_p2p`	True if the VIF is a point-to-point interface
`is_loopback`	True if the VIF is a loopback interface
`is_multicast`	True if the VIF is a multicast interface
`is_broadcast`	True if the VIF is a broadcast interface

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

< 0 on other errors

HAL_SUCCESS when function succeeds

hal_ipv4_mc_set_min_ttl_threshold

This function sets the minimum TTL (time-to-live) a multicast packet must have to be forwarded on this virtual interface.

Syntax

```
int
hal_ipv4_mc_set_min_ttl_threshold (unsigned int ifindex, unsigned char ttl);
```

Input Parameters

`index`	VIF index
`ttl`	TTL threshold

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv4_mc_get_min_ttl_threshold

This function gets the minimum TTL a multicast packet is to be forwarded on this virtual interface.

Syntax

```
int
hal_ipv4_mc_get_min_ttl_threshold (unsigned int index, unsigned char ttl);
```

Input Parameters

`index`	VIF index
`ttl`	TTL threshold

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv4_mc_set_max_rate_limit

This function sets the maximum multicast bandwidth rate allowed on this virtual interface.

Syntax

```
int
hal_ipv4_mc_set_max_rate_limit (unsigned int index,
                                unsigned int rate_limit);
```

Input Parameters

`index`	VIF index
`rate_limit`	Rate limit

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv4_mc_get_max_rate_limit

This function gets the maximum multicast bandwidth rate allowed on this virtual interface.

Syntax

```
int
hal_ipv4_mc_get_max_rate_limit (unsigned int index,
                                unsigned int rate_limit);
```

Input Parameters

`index`	VIF index
`rate_limit`	Rate limit

Output Parameters

None

Return Value

HAL_IPV4_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv4_mc_add_mfc

This function installs or modifies a multicast forwarding cache (MFC). If the MFC entry; source, group (S, G); is not found; a new one is created; otherwise, the existing entry is modified.

Syntax

```
int
hal_ipv4_mc_add_mfc (struct hal_in4_addr *source, struct hal_in4_addr *group,
                     u_int32_t iif_vif_index, int num_oifs,
                     u_int16_t *olist_vifs, u_char *oifs_ttl)
```

Input Parameters

`source`	Source address
`group`	Group address
`iif_vif_index`	MFC incoming interface index
`num_ttls`	Number of elements in TTL array
`olist_vifs`	
`oifs_ttl`	An array with the minimum Time to Live (TTL) a packet should be forwarded

Output Parameters

None

Return Value

HAL_IPV4_MC_MFC

HAL_SUCCESS when function succeeds

hal_ipv4_mc_delete_mfc

This function deletes an MFC entry.

Syntax

```
int
hal_ipv4_mc_delete_mfc (struct hal_in4_addr *source, struct hal_in4_addr *group,
                        u_int32_t index)
```

Input Parameters

`source`	Source address
`group`	Group address
`index`	VIF index

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv4_mc_get_sg_count

This function gets the various counters per (S, G) entry.

Syntax

```
int
hal_ipv4_mc_get_sg_count (struct hal_in4_addr *source,
                          struct hal_in4_addr *group,
                          u_int32_t iif_vif,
                          u_int32_t *pktcnt,
                          u_int32_t *bytecnt,
                          u_int32_t *wrong_vif);
```

Input Parameters

`src`	Source address
`group`	Group address
`pktcnt`	Packet count
`bytecnt`	Byte count
`wrong_vif`	Wrong VIFs

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

IPv6 API

The following subsection describes the Pv6 functions.

Table 2-9: IPv6 API Functions

Functions	Description
hal_ipv6_init	This function initializes the IPv6 hardware layer component.
hal_ipv6_deinit	This function deinitializes the IPv6 hardware layer component.
hal_ipv6_nbr_add	This function adds an IPv6 neighbor entry.
hal_ipv6_nbr_del	This function deletes an IPv6 neighbor entry.
hal_ipv6_nbr_cache_get	This function deletes all IPv6 neighbor entries.
hal_ipv6_uc_init	This function initializes the IPv6 unicast table for the specified FIB ID.
hal_ipv6_uc_deinit	This function deinitializes the IPv6 unicast table for the specified FIB ID.
hal_ipv6_uc_route_add	This function adds an IPv6 unicast route to the forwarding plane.
hal_ipv6_uc_route_delete	This function deletes an IPv6 unicast route from the forwarding plane.
hal_ipv6_uc_route_update	This function updates an IPv6 unicast route for the forwarding plane.
hal_ipv6_mc_init	This function initializes the IPv6 multicast table for the specified FIB ID.
hal_ipv6_mc_deinit	This function deinitializes the IPv6 multicast table for the specified FIB ID.
hal_ipv6_mc_pim_init	This function initializes PIM routing for the specified FIB ID.
hal_ipv6_mc_pim_deinit	This function deinitializes PIM routing for the specified FIB ID.
hal_ipv6_mc_get_max_vifs	This function gets the maximum number of VIFs supported.
hal_ipv6_mc_vif_add	This function creates a VIF.
hal_ipv6_mc_vif_delete	This function deletes a VIF.
hal_ipv6_mc_vif_addr_add	This function adds an address to a configured VIF.
hal_ipv6_mc_vif_addr_delete	This function deletes an address from a configured VIF.
hal_ipv6_mc_vif_set_physical_if	This function sets the physical interface index to a configured VIF.

Table 2-9: IPv6 API Functions

Functions	Description
hal_ipv6_mc_vif_set_flags	This function sets the VIF flags of a configured VIF.
hal_ipv6_mc_set_min_ttl_threshold	This function sets the minimum TTL a multicast packet requires to be forwarded.
hal_ipv6_mc_get_min_ttl_threshold	This function gets the minimum TTL a multicast packet will have to be forwarded.
hal_ipv6_mc_set_max_rate_limit	This function sets the maximum multicast bandwidth rate allowed on a VIF.
hal_ipv6_mc_get_max_rate_limit	This function gets the maximum multicast bandwidth rate allowed on a VIF.
hal_ipv6_mc_add_mfc	This function adds the maximum multicast bandwidth rate.
hal_ipv6_mc_delete_mfc	This function deletes an MFC entry.
hal_ipv6_mc_get_sg_count	This function gets the various counters per (S, G) entry.

hal_ipv6_init

This function initializes the IPv6 hardware layer component.

Syntax

```
int
hal_ipv6_init (void)
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_deinit

This function deinitializes the IPv6 hardware layer component.

Syntax

```
int
hal_ipv6_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_nbr_add

This function adds an IPv6 neighbor entry.

Syntax

```
int
hal_ipv6_nbr_add(struct pal_in6_addr *addr,
                  unsigned char *mac_addr,
                  u_int32_t ifindex)
```

Input Parameters

addr	IPv6 address
mac_addr	MAC address
ifindex	Interface index

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_nbr_del

This function deletes an IPv6 neighbor entry.

Syntax

```
int
hal_ipv6_nbr_del (struct pal_in6_addr *addr,
                  unsigned char *mac_addr,
                  unsigned int ifindex)
```

Input Parameters

addr	IPv6 address
mac_addr	MAC address
ifindex	Interface index

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_nbr_del_all

This function deletes all IPv6 neighbor entries, whether they are dynamic or static.

Syntax

```
int
hal_ipv6_nbr_del_all (unsigned short fib_id, u_char clr_flag)
```

Input Parameters

fib_id	FIB table ID
clr_flag	Flag to indicate dynamic or static ARP entries

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_nbr_cache_get

This function gets the IPv6 neighbor table starting at the next address of the IPv6 address, and the count number of entries. It returns the actual number of entries found as the return parameter. It is expected the memory is allocated by the caller before calling this API. To get the entire table, use null for the IPv6 address.

Syntax

```
int
hal_ipv6_nbr_cache_get(unsigned short fib_id,
                       struct pal_in6_addr *addr,
                       int count,
                       struct hal_ipv6_nbr_cache_entry *cache)
```

Input Parameters

fib_id	FIB Table ID
addr	IPv6 address to get the IPv6 neighbor table starting at the next address of the IPv6 address; null to get the entire table
count	Number of entries
cache	IPv6 neighbor cache

Output Parameters

None

Return Value

Number of entries. Can be 0 for no entries.

hal_ipv6_uc_init

This function initializes the IPv6 unicast table for the specified FIB ID.

Syntax

```
int
hal_ipv6_uc_init (unsigned short fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_uc_deinit

This function deinitializes the IPv6 unicast table for the specified FIB ID.

Syntax

```
int
hal_ipv6_uc_deinit (unsigned short fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_uc_route_add

This function adds an IPv6 unicast route to the forwarding plane.

Syntax

```
int
hal_ipv6_uc_route_add (unsigned short fib,
                       struct pal_in6_addr *ipaddr,
                       unsigned char ipmask,
                       unsigned short num, struct hal_ipv6uc_nexthop *nexthops)
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`ipmask`	IP mask length
`num`	Number of nexthops
`nexthops`	List of nexthops

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_uc_route_delete

This function deletes an IPv6 route from the forwarding plane.

Syntax

```
int
hal_ipv6_uc_route_delete (unsigned short fib,
                          struct pal_in6_addr *ipaddr,
                          unsigned char ipmask,
                          int num, struct hal_ipv6uc_nexthop *nexthops)
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`ipmask`	IP mask length
`num`	Number of nexthops
`nexthops`	List of nexthops

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_uc_route_update

This function updates an existing IPv6 route with the new nexthop parameters.

Syntax

```
int
hal_ipv6_uc_route_update (unsigned short fib,
                          struct pal_in6_addr *ipaddr,
                          unsigned char ipmask,
                          int numfib, struct hal_ipv6uc_nexthop *nexthopsfib,
                          int numnew, struct hal_ipv6uc_nexthop *nexthopsnew);
```

Input Parameters

`fib`	FIB ID
`ipaddr`	IP address
`ipmask`	IP mask length
`numfib`	Number of nexthops in the FIB
`nexthopsfib`	List of nexthops in the FIB
`numnew`	Number of new or updated nexthops
`nexthopsnew`	List of new or updated nexthops in the FIB

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_init

This function initializes the IPv6 multicast table for the specified FIB ID.

Syntax

```
int
hal_ipv6_mc_init (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_deinit

This function deinitializes the IPv6 multicast table for the specified FIB ID.

Syntax

```
int
hal_ipv6_mc_deinit (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_pim_init

This function starts PIM routing for the specified FIB ID.

Syntax

```
int
hal_ipv6_mc_pim_init (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_pim_deinit

This function stops PIM routing for the specified FIB ID.

Syntax

```
int
hal_ipv6_mc_pim_deinit (int fib);
```

Input Parameters

`fib`	FIB ID

Output Parameters

None

Return Value

HAL_IP_FIB_NOT_EXIST

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_get_max_vifs

This function gets the maximum number of VIFs supported.

Syntax

```
int
hal_ipv6_mc_get_max_vifs (int *vifs);
```

Input Parameters

`vifs`	Maximum VIFs

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_add

This function creates a VIF.

Syntax

```
int
hal_ipv6_mc_vif_add (u_int32_t vif_index, u_int32_t phy_ifindex,
    u_int16_t flags)
```

Input Parameters

`vif_index`	VIF index
`phy_index`	Physical Interface index
`flags`	VIF type

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_EXISTS

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_delete

This function deletes a VIF.

Syntax

```
int
hal_ipv6_mc_vif_delete (u_int32_t vif_index);
```

Input Parameters

`index`	VIF index

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_IPV6_MC_VIF_MAX_EXCEEDED

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_addr_add

This function adds an address to a configured VIF.

Syntax

```
int
hal_ipv6_mc_vif_addr_add (unsigned int index,
                          struct pal_in6_addr *addr,
                          struct pal_in6_addr *subnet,
                          struct pal_in6_addr *broadcast,
                          struct pal_in6_addr *peer)
```

Input Parameters

`index`	VIF index
`addr`	Address to add
`subnet`	Subnet address to add
`broadcast`	Broadcast address to add

`peer`	Peer address to add

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_addr_delete

This function deletes an address from a configured VIF.

Syntax

```
int
hal_ipv6_mc_vif_addr_delete (unsigned int index, struct pal_in6_addr *addr);
```

Input Parameters

`index`	VIF index
`addr`	Address to delete

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_IPV6_MC_VIF_ADDRESS_NOTFOUND

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_set_physical_if

This function sets the physical interface index to a configured VIF.

Syntax

```
int
hal_ipv6_mc_vif_set_physical_if (unsigned int index, unsigned int ifindex);
```

Input Parameters

`index`	VIF index
`ifindex`	Physical interface index

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_IPV6_MC_IF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_vif_set_flags

This function sets the VIF flags of a configured VIF.

Syntax

```
int
hal_ipv6_mc_vif_set_flags (unsigned int ifindex,
                           unsigned char is_pim_register,
                           unsigned char is_p2p,
                           unsigned char is_loopback,
                           unsigned char is_multicast,
                           unsigned char is_broadcast)
```

Input Parameters

`ifindex`	VIF index
`is_pim_register`	True if the VIF is a PIM register interface
`is_p2p`	True if the VIF is a point-to-point interface
`is_loopback`	True if the VIF is a loopback interface
`is_multicast`	True if the VIF is a multicast interface
`is_broadcast`	True if the VIF is a broadcast interface

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

< 0 on other errors

HAL_SUCCESS when function succeeds

hal_ipv6_mc_set_min_ttl_threshold

This function sets the minimum TTL a multicast packet requires to be forwarded on this virtual interface.

Syntax

```
int
hal_ipv6_mc_set_min_ttl_threshold (unsigned int ifindex, unsigned char ttl);
```

Input Parameters

`index`	VIF index
`ttl`	TTL threshold

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXITS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_get_min_ttl_threshold

This function gets the minimum TTL a multicast packet will have to be forwarded on this virtual interface.

Syntax

```
int
hal_ipv6_mc_get_min_ttl_threshold (unsigned int index, unsigned char ttl);
```

Input Parameters

`index`	VIF index
`ttl`	TTL threshold

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_set_max_rate_limit

This function sets the maximum multicast bandwidth rate allowed on a VIF.

Syntax

```
int
hal_ipv6_mc_set_max_rate_limit (unsigned int index,
                                unsigned int rate_limit)
```

Input Parameters

`index`	VIF index
`rate_limit`	Rate limit

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_get_max_rate_limit

This function gets the maximum multicast bandwidth rate allowed on a VIF.

Syntax

```
int
hal_ipv6_mc_get_max_rate_limit (unsigned int index,
                                unsigned int rate_limit)
```

Input Parameters

`index`	VIF index
`rate_limit`	Rate limit

Output Parameters

None

Return Value

HAL_IPV6_MC_VIF_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_add_mfc

This function installs or modifies an MFC. If the MFC entry, source, group (S, G), is not found; a new one is created; otherwise, the existing entry is modified.

Syntax

```
int
hal_ipv6_mc_add_mfc (struct hal_in6_addr *source, struct hal_in6_addr *group,
                     u_int32_t iif_vif_index, u_int32_t num_olist,
                     u_int16_t *olist)
```

Input Parameters

`source`	Source address
`group`	Group address
`iif_vif_index`	MFC incoming interface index
`num_olist`	Number of unsigned 32-bit elements in the olist bitmap
`olist`	Array of VIF indices in the olist

Output Parameters

None

Return Value

HAL_IPV6_MC_MFC

HAL_SUCCESS when function succeeds

hal_ipv6_mc_delete_mfc

This function deletes an MFC entry.

Syntax

```
iint
hal_ipv6_mc_delete_mfc (struct hal_in6_addr *source, struct hal_in6_addr *group)
```

Input Parameters

`source`	Source address
`group`	Group address

Output Parameters

None

Return Value

HAL_IPV6_MC_MFC_NOTEXISTS

HAL_SUCCESS when function succeeds

hal_ipv6_mc_get_sg_count

This function gets the various counters per (S, G) entry.

Syntax

```
int
hal_ipv6_mc_get_sg_count (struct hal_in6_addr *source,
                          struct hal_in6_addr *group,
                          u_int32_t iif_vif,
                          u_int32_t *pktcnt,
                          u_int32_t *bytecnt,
                          u_int32_t *wrong_vif)
```

Input Parameters

`source`	Source address
`group`	Group address
`iif_vif`	Incoming VIF
`pktcnt`	Packet count
`bytecnt`	Byte count
`wrong_vif`	Wrong VIFs

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

Link Aggregation Control Protocol API

The following subsection includes the Link Aggregation Control Protocol (LACP) functions.

Table 2-10: LACP API Functions

Functions	Description
hal_lacp_init	This function initializes the LACP hardware layer component.
hal_lacp_deinit	This function deinitializes the LACP hardware layer component.
hal_lacp_add_aggregator	This function adds an aggregator with a name and MAC address.
hal_lacp_delete_aggregator	This function deletes an aggregator.
hal_lacp_attach_mux_to_aggregator	This function adds a port to an aggregator.
hal_lacp_psc_set	This function sets load-balancing mode for an aggregator.
hal_lacp_collecting	This function enables or disables collecting on a port.
hal_lacp_distributing	This function enables or disables distributing for a port.
hal_lacp_collecting_distributing	This function enables or disables distributing for, and collecting on, a port.

hal_lacp_init

This function initializes the LACP (Link Aggregation Control Protocol) hardware layer component.

Syntax

```
int
hal_lacp_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_LACP_INIT when function fails

HAL_SUCCESS when function succeeds

hal_lacp_deinit

This function deinitializes the link aggregation (LACP) hardware layer component.

Syntax

```
int
```

```
hal_lacp_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_LACP_DEINIT

HAL_SUCCESS when function succeeds

hal_lacp_add_aggregator

This function adds an aggregator with the specified name and MAC address.

Syntax

```
int
hal_lacp_add_aggregator (char *name, unsigned char mac[], int agg_type);
```

Input Parameters

`name`	Aggregator name
`mac`	MAC address of aggregator
`agg_type`	Aggregator type (L2/L3)

Output Parameters

None

Return Value

HAL_ERR_LACP_EXISTS

HAL_SUCCESS when function succeeds

hal_lacp_delete_aggregator

This function deletes an aggregator.

Syntax

```
int
hal_lacp_delete_aggregator (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Aggregator name
`ifindex`	Aggregator interface index

Output Parameters

None

Return Value

HAL_ERR_LACP_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_lacp_attach_mux_to_aggregator

This function adds a port to an aggregator.

Syntax

```
int
hal_lacp_attach_mux_to_aggregator (char *agg_name, unsigned int agg_ifindex,
                                   char *port_name, unsigned int port_ifindex);
```

Input Parameters

`agg_name`	Aggregator name
`agg_ifindex`	Aggregator interface index
`port_name`	Name of the port
`port_ifindex`	Port interface index

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_lacp_detach_mux_from_aggregator

This function deletes a port from an aggregator.

Syntax

```
int
hal_lacp_detach_mux_from_aggregator (char *agg_name, unsigned int agg_ifindex,
                                     char *port_name, unsigned int port_ifindex);
```

Input Parameters

`agg_name`	Aggregator name
`agg_ifindex`	Aggregator interface index
`port_name`	Name of the port
`port_ifindex`	Port interface index

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_lacp_psc_set

This function sets load-balancing mode for an aggregator.

Syntax

```
int
hal_lacp_psc_set (unsigned int ifindex,int psc);
```

Input Parameters

ifindex	Aggregator interface index
psc	Port selection criteria (source MAC or destination MAC based)

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_lacp_collecting

This function enables or disables collecting on a port.

Syntax

```
int
hal_lacp_collecting (char *name, unsigned int ifindex, int enable);
```

Input Parameters

name	Aggregator name
ifindex	Port interface index
enable	1 Enables collecting
	0 Disables collecting

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_lacp_distributing

This function enables or disables distributing for a port.

Syntax

```
int
hal_lacp_distributing (char *name, unsigned int ifindex, int enable);
```

Input Parameters

`name`	Aggregator name
`ifindex`	Port interface index
`enable`	`1` Enables collecting
	`0` Disables collecting

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

hal_lacp_collecting_distributing

This function enables or disables distributing for, and collecting on, a port.

Syntax

```
int
hal_lacp_collecting_distributing (char *name, unsigned int ifindex, int enable);
```

Input Parameters

`name`	Aggregator name
`ifindex`	Port interface index
`enable`	`1` Enables distributing and collecting
	`0` Disables distributing and collecting

Output Parameters

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

Layer 2 Forwarding Database API

The following subsection includes the layer 2 forwarding database functions.

Table 2-11: Layer 2 Forwarding Database API Functions

Functions	Description
hal_l2_fdb_init	This function initializes the layer 2 Forwarding Database (FDB) table.
hal_l2_fdb_deinit	This function deinitializes the layer 2 Forwarding Database (FDB) table.
hal_l2_add_fdb	This function adds a layer 2 forwarding database (FDB) entry.
hal_l2_del_fdb	This function deletes a layer 2 forwarding database (FDB) entry.
hal_l2_fdb_unicast_get	This function gets the unicast HAL FDB entry.
hal_l2_fdb_multicast_get	This function gets the multicast HAL FDB entry
hal_l2_add_priority_ovr	This function adds a L2 FDB entry with priority override entry.
hal_l2_bcast_discards_get	This function gets the number of discarded broadcast frames.
hal_l2_mcast_discards_get	This function gets the number of discarded multicast frames.
hal_l2_dlf_bcast_discards_get	This function gets the number of discarded destination lookup failures.

hal_l2_fdb_init

This function initializes the layer 2 Forwarding Database (FDB) table.

Syntax

```
int
hal_l2_fdb_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_L2_FDB_INIT when function fails

HAL_SUCCESS when function succeeds

hal_l2_fdb_deinit

This function deinitializes the layer 2 Forwarding Database (FDB) table.

Syntax

```
int
hal_l2_fdb_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_L2_FDB_INIT when function fails

HAL_SUCCESS when function succeeds

hal_l2_add_fdb

This function adds a layer 2 forwarding database (FDB) entry. If the flag parameter is set to HAL_L2_FDB_STATIC, the entry is added as static, and will not age. If the flag parameter is 0, the entry is added as a dynamic entry, and will age.

Syntax

```
int
hal_l2_add_fdb (const char * const name, unsigned int ifindex,
                const unsigned char * const mac, int len,
                unsigned short vid, unsigned short svid,
                unsigned char flags, bool_t is_forward);
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index
`vid`	VLAN ID
`svid`	S-VLAN ID
`mac`	MAC address
`len`	MAC address length
`flags`	

Output Parameters

None

Return Value

HAL_ERR_L2_FDB_ENTRY_EXISTS

HAL_ERR_L2_FDB_ENTRY

HAL_SUCCESS when function succeeds

hal_l2_del_fdb

This function deletes a layer 2 forwarding database (FDB) entry. If flag is set to HAL_L2_FDB_STATIC, the entry is added as static, and will not age. If the flag parameter is 0, the entry is added as a dynamic entry, and will age.

Syntax

```
int
hal_l2_del_fdb (const char * const name, unsigned int ifindex,
                const unsigned char * const mac, int len,
                unsigned short vid, unsigned short svid,
                unsigned char flags)
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index
`vid`	VLAN ID
`svid`	S-VLAN ID
`mac`	MAC address
`len`	MAC address length
`flags`	

Output Parameters

None

Return Value

HAL_ERR_L2_FDB_NO_ENTRY

HAL_ERR_L2_FDB_ENTRY

HAL_SUCCESS when function succeeds

hal_l2_fdb_unicast_get

This function gets the unicast HAL FDB entry starting at the next address of the MAC address and specified VLAN ID. This function gets the count number of entries. Returns the actual number of entries found as the return parameter. It is expected that the memory is allocated by the caller before calling this API.

Syntax

```
int
hal_l2_fdb_unicast_get (char *name, char *mac_addr, unsigned short vid,
                        u_int16_t count, struct hal_fdb_entry *fdb_entry)
```

Input Parameters

`name`	Bridge name
`mac_addr`	FDB entry after this MAC address
`vid`	FDB entry after this VLAN ID
`count`	Number of FDB entries to be returned

Output Parameters

`fdb_entry`	Array of FDB entries returned

Return Value

HAL_ERR_L2_FDB_ENTRY number of entries. Can be 0 for no entries.

hal_l2_fdb_multicast_get

This function gets the multicast HAL FDB entry starting at the next address of the MAC address and specified VLAN ID. This function gets the count number of entries. Returns the actual number of entries found as the return parameter. It is expected that the memory is allocated by the caller before calling this API.

Syntax

```
int
hal_l2_fdb_multicast_get (char *name, char *mac_addr, unsigned short vid,
                          u_int16_t count, struct hal_fdb_entry *fdb_entry)
```

Input Parameters

`name`	Bridge name
`mac_addr`	FDB entry after this MAC address
`vid`	FDB entry after this VLAN ID
`count`	Number of FDB entries to be returned

Output Parameters

`fdb_entry`	Array of FDB entries returned

Return Value

HAL_ERR_L2_FDB_ENTRY number of entries. Can be 0 for no entries.

hal_l2_add_priority_ovr

This function adds a L2 FDB entry with priority override entry.

Syntax

```
int
hal_l2_add_priority_ovr (const char * const name, unsigned int ifindex,
                         const unsigned char * const mac, int len,
                         unsigned short vid,
                         unsigned char ovr_mac_type,
                         unsigned char priority)
```

Input Parameters

`name`	Bridge name
`ifindex`	Port interface index
`mac`	MAC address
`len`	MAC address length

`vid`	VLAN ID
`ovr_mac_type`	Type of ATU entry
`priority`	Priority

Output Parameters

None

Return Value

HAL_L2_FDB_ENTRY_EXISTS

HAL_L2_FDB_ENTRY

HAL_SUCCESS when function succeeds

hal_l2_bcast_discards_get

This function gets the number of discarded broadcast frames.

Syntax

```
int
hal_l2_bcast_discards_get (unsigned int ifindex,
                            unsigned int *discards);
```

Input Parameters

`ifindex`	Port interface index

Output Parameters

`discards`	Number of discarded frames

Return Value

HAL_ERR_RATELIMIT_BCAST

HAL_SUCCESS when function succeeds

hal_l2_mcast_discards_get

This function gets the number of discarded multicast frames.

Syntax

```
int
hal_l2_mcast_discards_get (unsigned int ifindex,
                            unsigned int *discards);
```

Input Parameters

`ifindex`	Port interface index

Output Parameters

`discards`	Number of discarded frames

Return Value

HAL_ERR_RATELIMIT_MCAST

HAL_SUCCESS when function succeeds

hal_l2_dlf_bcast_discards_get

This function gets the number of discarded destination lookup failures.

Syntax

```
int
hal_l2_dlf_bcast_discards_get (unsigned int ifindex,
                               unsigned int *discards);
```

Input Parameters

ifindex	Port interface index

Output Parameters

discards	Number of discarded frames

Return Value

HAL_ERR_RATELIMIT_MCAST

HAL_SUCCESS when function succeeds

MPLS API

The following are the MPLS functions.

Table 2-12: MPLS API Functions

Functions	Description
hal_mpls_init	This function initializes the MPLS forwarding plane.
hal_mpls_deinit	This function deinitializes the MPLS forwarding plane.
hal_mpls_vrf_create	This function creates a VRF table.
hal_mpls_vrf_destroy	This function deletes a VRF table.
hal_mpls_enable_interface	This function enables an IP interface for MPLS forwarding.
hal_mpls_disable_interface	This function disables interface for MPLS forwarding.
hal_mpls_if_update_vrf	This function updates the VRF to which the MPLS interface points.
hal_mpls_clear_fib_table	This function clears all MPLS FIB entries matching a specified identifier.
hal_mpls_clear_vrf_table	This function clears all VRF entries matching a specified identifier.
hal_mpls_ftn_entry_add	This function adds the specified FTN entry to the FTN table.

Table 2-12: MPLS API Functions

Functions	Description
hal_mpls_ftn_entry_delete	This function removes the specified entry from the FTN table.
hal_mpls_ilm_entry_add	This function adds the specified ILM entry to the ILM table.
hal_mpls_ilm_entry_delete	This function deletes a specified ILM entry to the ILM table.
hal_mpls_send_ttl	This function sets the new TTL value for all packets.
hal_mpls_local_pkt_handle	This function enables or disables the mapping of locally generated packets.
hal_mpls_vc_init	This function binds an interface to a virtual circuit.
hal_mpls_vc_deinit	This function unbinds an interface from a virtual circuit.
hal_mpls_vc_fib_add	This function adds a VC FIB (FTN/ILM) entry for a VC peer.
hal_mpls_vc_fib_delete	This function deletes a VC FIB entry (FTN/ILM).
hal_mpls_vpls_add	This function adds a VPLS entry.
hal_mpls_vpls_del	This function deletes a VPLS entry.
hal_mpls_vpls_if_bind	This function binds a VPLS entry to an interface.
hal_mpls_vpls_if_unbind	This function unbinds a VPLS entry to an interface.
hal_mpls_qos_reserve	This function reserves MPLS QoS resources after creation of the LSP.
hal_mpls_qos_release	This function releases MPLS QoS resources after tear down of the LSP.

hal_mpls_init

This function initializes the MPLS forwarding plane.

Syntax

```
int
hal_mpls_init (u_char protocol);
```

Input Parameters

protocol	Protocol identifier

Output Parameters

None

Return Value

HAL_MPLS_INIT_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_deinit

This function de-initializes the MPLS forwarding plane.

Syntax

```
int
hal_mpls_deinit (u_char protocol);
```

Input Parameters

`protocol`	Protocol identifier

Output Parameters

None

Return Value

HAL_MPLS_DEINIT_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vrf_create

This function creates a VRF (Virtual Routing and Forwarding) table.

Syntax

```
int
hal_mpls_vrf_create (int vrf);
```

Input Parameters

`vrf`	VRF table ID

Output Parameters

None

Return Value

HAL_MPLS_VRF_EXISTS

HAL_SUCCESS when function succeeds

hal_mpls_vrf_destroy

This function deletes a VRF table.

Syntax

```
int
hal_mpls_vrf_destroy (int vrf);
```

Input Parameters

`vrf`	VRF table ID

Output Parameters

None

Return Value

HAL_MPLS_VRF_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_mpls_enable_interface

This function enables an IP interface for MPLS forwarding. If an interface is already MPLS-enabled, then this API can be used to change the association of the label space. The new label space is bound to this interface. A new ILM table is created for a new label space identifier.

Syntax

```
int
hal_mpls_enable_interface (struct if_ident *if_ident,
                           unsigned short label_space)
```

Input Parameters

`if_ident`	Interface identifier
`label_space`	Label space identifier

Output Parameters

None

Return Value

HAL_MPLS_INTERFACE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_disable_interface

This function disables interface for MPLS forwarding. If the reference count of the ILM table to which this interface is bound becomes 0, then the ILM table is deleted.

Syntax

```
int
hal_mpls_disable_interface (struct if_ident *if_ident)
```

Input Parameters

`if_ident`	Interface identifier

Output Parameters

None

Return Value

HAL_MPLS_INTERFACE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_if_update_vrf

This function updates the VRF to which the MPLS interface points.

Syntax

```
int
hal_mpls_if_update_vrf (struct if_ident *if_ident, int vrf)
```

Input Parameters

`if_ident`	Interface identifier
`vrf`	VRF table identifier

Output Parameters

None

Return Value

HAL_MPLS_VRF_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_clear_fib_table

This function clears all the MPLS FIB entries matching the specified identifier. The identifier is application specific. For example, the applications can use a protocol identifier as an identifier for clearing out entries.

Syntax

```
int
hal_mpls_clear_fib_table (u_char protocol);
```

Input Parameters

`protocol`	Identifier for this entry

Output Parameters

None

Return Value

HAL_MPLS_CLEAR_FIB_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_clear_vrf_table

This function clears all the VRF entries matching the specified identifier. The identifier is application-specific. For example, the applications can use a protocol identifier as an identifier for clearing out entries.

Syntax

```
int
```

```
hal_mpls_clear_vrf_table (u_char protocol);
```

Input Parameters

`protocol`	Identifier for this entry

Output Parameters

None

Return Value

HAL_MPLS_CLEAR_VRF_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_ftn_entry_add

This function adds the specified FTN entry to the FTN table. If the entry exists, this request is ignored. This function can also be used to modify an existing entry.

Syntax

```
int
hal_mpls_ftn_entry_add (int vrf,
                        u_char protocol,
                        struct hal_in4_addr *fec_addr,
                        u_char *fec_prefix_len,
                        u_char *dscp_in,
                        u_int32_t *tunnel_label,
                        struct hal_in4_addr *tunnel_nhop,
                        struct if_ident *tunnel_if_ident,
                        u_int32_t *vpn_label,
                        struct hal_in4_addr *vpn_nhop,
                        struct if_ident *vpn_if_ident,
                        u_int32_t *tunnel_id,
                        u_int32_t *qos_resource_id,
#ifdef HAVE_DIFFSERV
                        struct hal_mpls_diffserv *diffserv_info,
#endif /* HAVE_DIFFSERV */
                        char opcode,
                        u_int32_t nhlfe_ix,
                        u_int32_t ftn_ix,
                        u_char ftn_type,
                        struct mpls_owner_fwd *owner,
                        u_int32_t bypass_ftn_ix,
                        u_char lsp_type)
```

Input Parameters

`vrf` VRF to which to add the entry

Note: A value of `HAL_MPLS_GLOBAL_FTN_TABLE` table adds the entry to the global FTN table.

`protocol` Identifier for this entry

`fec_addr` IP address of the FEC corresponding to this FTN entry `fec_prefix_len` Length of the prefix for this FEC

`dscp_in` DSCP code point

`tunnel_label` Tunnel LSP label. Only the lower order 20 bits are used. This is the LSP label or tunnel label used to carry layer 2/layer 3 VPN labels.

`tunnel_nexthop_addr`
IP address of the tunnel LSP next-hop to be used for this FEC

`tunnel_nexthop_if`
Nightspot interface for the tunnel LSP

`vpn_label` Inner label (VC/VRF label). Only the lower order 20 bits are used.

`vpn_nexthop_addr`
IP address of the VPN (VC/VRF) peer (only required for VC/VRF LSPs)

`vpn_outgoing_if`
Outgoing interface used for VPN (VC/VRF) peer. Optional parameter. Only required for VC/VRF LSPs. May be set to NULL.

`tunnel_id` Tunnel ID

`qos_resource_id` QoS resource ID

`tunnel_ds_info` Diffserv information for this FTN entry

`opcode` Opcode to be applied to this FTN entry (`HAL_MPLS_PUSH`, `HAL_MPLS_PUSH_AND_LOOKUP`, `HAL_MPLS_DLVR_TO_IP`)

`nhlfe_ix` Next-hop label forwarding entry index

`ftn_ix` FTN index

`ftn_type` FTN type

`owner` MPLS owner type

`bypass_ftn_ix` Bypass FTN index

`lsp_type` LSP type; value may be one of the following:

`LSP_TYPE_PRIMARY`	1
`LSP_TYPE_SECONDARY`	2
`LSP_TYPE_BACKUP`	3
`LSP_TYPE_BYPASS`	4

Note: If the forwarder does not support multiple entries (only the primary LSP type is supported), multiple LSP types can be turned off. To do this, change `define HAVE_MPLS_INSTALL_BK_LSP` to `undef HAVE_MPLS_INSTALL_BK_LSP` in the `nsm_mpls.c` file.

Output Parameters

None

Return Value

HAL_MPLS_FTN_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_ftn_entry_delete

This function removes the specified entry from the FTN table. If the identifier does not match the one stored in the FTN entry, the delete operation will fail.

Syntax

```
int
hal_mpls_ftn_entry_delete (int vrf,
                           u_char protocol,
                           struct hal_in4_addr *fec_addr,
                           u_char *fec_prefix_len,
                           u_char *dscp_in,
                           struct hal_in4_addr *tunnel_nhop,
                           u_int32_t nhlfe_ix,
                           u_int32_t *tunnel_id,
                           u_int32_t ftn_ix)
```

Input Parameters

`vrf`	VRF table identifier
`protocol`	Entry identifier
`fec_addr`	IP address of FEC corresponding to this FTN entry
`fec_prefix_len`	Length of the prefix for this FEC
`dscp_in`	DSCP code point
`tunnel_nexthop`	IP address of the tunnel LSP next-hop to be used for this FEC
`nhlfe_ix`	Next-hop label forwarding entry index
`tunnel_id`	Tunnel ID
`ftn_ix`	FTN index

Output Parameters

None

Return Value

HAL_MPLS_FTN_DELETE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_ilm_entry_add

This function adds the specified ILM entry to the ILM table. If this entry already exists in the ILM table, the request is ignored. This function can also be used to modify an existing entry.

Syntax

```
int
hal_mpls_ilm_entry_add (u_int32_t *in_label,
                        struct if_ident *in_if,
                        u_char opcode,
                        struct hal_in4_addr *nexthop,
                        struct if_ident *out_if,
                        u_int32_t *swap_label,
                        u_int32_t nhlfe_ix,
                        u_char is_egress,
                        u_int32_t *tunnel_label,
                        u_int32_t *qos_resource_id,
#ifdef HAVE_DIFFSERV
                        struct hal_mpls_diffserv *ds_info,
#endif /* HAVE_DIFFSERV */
                        struct hal_in4_addr *fec_addr,
                        unsigned char *fec_prefixlen,
                        u_int32_t vpn_id,
                        struct hal_in4_addr *vc_peer);
```

Input Parameters

in_label	Incoming label ID. Only the low-order 20 bits are used.
in_if	Identifying object for the incoming interface
opcode	Operation code to be applied for this FTN entry (HAL_MPLS_POP, HAL_MPLS_SWAP, HAL_MPLS_POP_FOR_VPN)
nexthop	IP address of the next-hop to be used for this FEC
out_if	Identifying object for the outgoing interface
swap_label	ID of the swap label. Only the low order 20 bits are used.
nhlfe_ix	Next-hop label forwarding entry index
is_egress	Flag to identify whether the LSR is a egress for this FEC.
tunnel_label	ID of the tunnel label (if any).
ds_info	Diffserv information for ILM entry
fec_addr	IP address of FEC corresponding to this ILM entry
fec_prefixlen	Length of the prefix for this FEC
vpn_id	VPI identifier
vc_peer	VC peer address (for VC ILM entries only)

Output Parameters

None

Return Value

HAL_MPLS_ILM_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_ilm_entry_delete

This function deletes the specified entry from the ILM table. If this entry is not present in the ILM table, this request is ignored. If the identifier does not match to the one stored in the ILM entry, the delete operation fails.

Syntax

```
int
```

```
hal_mpls_ilm_entry_delete (u_char protocol,
                           u_int32_t *label_id_in,
                           struct if_ident *if_info);
```

Input Parameters

`protocol`	Identifier for this ILM entry
`label_id_in`	Incoming label ID. Only the low-order 20 bits are used.
`if_ident`	Identifying object for the incoming interface

Output Parameters

None

Return Value

HAL_MPLS_ILM_DELETE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_send_ttl

This function sets the new TTL value for all packets that are switched through the LSPs that use the current LSR for either ingress or egress. A value of -1 for the new TTL uses the default mechanism (the copying of TTL from IP packet to labeled packet and vice-versa).

Syntax

```
int
hal_mpls_send_ttl (u_char protocol,
                   u_char type,
                   int ingress,
                   int new_ttl);
```

Input Parameters

`protocol`	Identifier for this entry
`type`	Type
`is_igress`	Is ingress
`new_ttl`	New TTL value

Output Parameters

None

Return Value

HAL_MPLS_TTL_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_local_pkt_handle

This function is used to enable or disable the mapping of locally generated packets.

Syntax

```
int
hal_mpls_local_pkt_handle (u_char protocol,
                           int enable);
```

Input Parameters

`protocol`	Identifier for this entry
`enable`	1 = enable, 0 = disable

Output Parameters

None

Return Value

HAL_MPLS_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vc_init

This function binds an interface to a Virtual Circuit.

Syntax

```
int
hal_mpls_vc_init (u_int32_t vc_id,
                  struct if_ident *if_info,
                  u_int16_t vlan_id);
```

Input Parameters

`vc_id`	Virtual Circuit identifier
`if_info`	Interface identifier
`vlan_id`	VLAN identifier

Output Parameters

None

Return Value

HAL_MPLS_VC_BIND_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vc_deinit

This function unbinds an interface from a Virtual Circuit (VC).

Syntax

```
int
hal_mpls_vc_deinit (u_int32_t vc_id,
                    struct if_ident *if_info,
                    u_int16_t vlan_id);
```

Input Parameters

`vc_id`	Virtual Circuit identifier
`if_info`	Interface identifier
`vlan_id`	VLAN identifier

Output Parameters

None

Return Value

HAL_MPLS_VC_UNBIND_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vc_fib_add

This function adds a VC FIB (FTN/ILM) entry for a VC peer. This function can be used to modify an existing FTN entry.

Syntax

```
int
hal_mpls_vc_fib_add (u_int32_t vc_id,
                       u_int32_t vc_style,
                       u_int32_t vpls_id,
                       u_int32_t in_label,
                       u_int32_t out_label,
                       u_int32_t ac_ifindex,
                       u_int32_t nw_ifindex,
                       u_char ftn_opcode,
                       struct pal_in4_addr *ftn_vc_peer,
                       struct pal_in4_addr *ftn_vc_nhop,
                       u_int32_t ftn_tunnel_label,
                       struct pal_in4_addr *ftn_tunnel_nhop,
                       u_int32_t ftn_tunnel_ifindex,
                       u_int32_t ftn_tunnel_nhlfe_ix);
```

Input Parameters

`vc_id`	Virtual Circuit identifier
`vc_style`	Type of VC (Mesh, Spoke, Martini)
`vpls_id`	VPLS Identifier
`in_label`	Incoming VC label
`out_label`	VC label to be pushed on the outgoing packet
`ac_ifindex`	Incoming interface index for incoming label
`nw_ifindex`	Outgoing interface to reach VC neighbor
`ftn_opcode`	MPLS opcode for VC FTN entry
`ftn_vc_peer`	Address of VC neighbor

`ftn_vc_nhop` Address of nexthop to reach VC neighbor

`ftn_tunnel_label` Tunnel label for carrying VC traffic

`ftn_tunnel_nhop` Address of nexthop node for tunnel LSP

`ftn_tunnel_ifindex` Outgoing interface index for tunnel LSP

`ftn_tunnel_nhlfe_ix` NHLFE index for tunnel FTN

Output Parameters

None

Return Value

HAL_MPLS_VPLS_FIB_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vc_fib_delete

This function deletes a VC FIB entry (FTN/ILM). If the identifier does not match, the delete operation fails.

Syntax

```
int
hal_mpls_vc_fib_delete (u_int32_t vc_id,
                        u_int32_t vc_style,
                        u_int32_t vpls_id,
                        u_int32_t in_label,
                        u_int32_t nw_ifindex,
                        struct hal_in4_addr *ftn_vc_peer);
```

Input Parameters

`vc_id` VC identifier

`vc_style` Type of VC (Mesh, Spoke)

`vpls_id` VPLS Identifier

`in_label` Incoming VC label

`nw_ifindex` Incoming interface index for incoming label

`ftn_vc_peer` Address of VC neighbor

Output Parameters

None

Return Value

HAL_MPLS_VPLS_FIB_DELETE_ERR when function fails

None

hal_mpls_vpls_add

This function adds a VPLS entry.

Syntax

```
int
hal_mpls_vpls_add (u_int32_t vpls_id);
```

Input Parameters

`vpls_id`	VPLS Identifier

Output Parameters

None

Return Value

HAL_MPLS_VPLS_FWD_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vpls_del

This function deletes a VPLS entry.

Syntax

```
int
hal_mpls_vpls_del (u_int32_t vpls_id);
```

Input Parameters

`vpls_id`	VPLS Identifier

Output Parameters

None

Return Value

HAL_MPLS_VPLS_FWD_ADD_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vpls_if_bind

This function binds a VPLS entry to an interface.

Syntax

```
int
hal_mpls_vpls_if_bind (u_int32_t vpls_id,
                       u_int32_t ifindex,
                       u_int16_t vlan_id);
```

Input Parameters

`vpls_id`	VPLS Identifier
`ifindex`	Interface index being bound to a VPLS
`vlan_id`	VLAN identifier

Output Parameters

None

Return Value

HAL_MPLS_VPLS_IF_BIND_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_vpls_if_unbind

This function unbinds a VPLS entry to an interface.

Syntax

```
int
hal_mpls_vpls_if_unbind (u_int32_t vpls_id,
                         u_int32_t ifindex,
                         u_int16_t vlan_id);
```

Input Parameters

`vpls_id`	VPLS Identifier
`ifindex`	Interface index being bound to a VPLS
`vlan_id`	VLAN identifier

Output Parameters

None

Return Value

HAL_MPLS_VPLS_IF_BIND_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_qos_reserve

This function reserves MPLS QoS resources after successful creation of the LSP.

Syntax

```
int
hal_mpls_qos_reserve (struct hal_mpls_qos *qos)
```

Input Parameters

`qos`	QoS-related parameter of the MPLS QoS resources.

Output Parameters

None

Return Value

HAL_MPLS_QOS_RESERVE_ERR when function fails

HAL_SUCCESS when function succeeds

hal_mpls_qos_release

This function releases MPLS QoS resources after tear down of the LSP.

Syntax

```
int
hal_mpls_qos_release (struct hal_mpls_qos *qos)
```

Input Parameters

qos	QoS-related parameter of the MPLS QoS resources.

Output Parameters

None

Return Value

HAL_MPLS_QOS_RELEASE_ERR when function fails

HAL_SUCCESS when function succeeds

Port Authentication Function API

The following subsection includes the port authentication API functions.

Table 2-13: HAL API Functions

Functions	Description
hal_auth_init	This function initializes the HAL for 802.1x port authentication.
hal_auth_deinit	This function deinitializes the HAL for 802.1x port authentication.
hal_auth_mac_set_port_state	This function sets the port auth_mac state.

hal_auth_init

This function initializes the hardware layer for 802.1x port authentication.

Syntax

```
int
hal_auth_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_AUTH_INIT when function fails

HAL_SUCCESS when function succeeds

hal_auth_deinit

This function deinitializes the hardware layer for 802.1x port authentication.

Syntax

```
int
hal_auth_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_AUTH_INIT when function fails

HAL_SUCCESS when function succeeds

hal_auth_mac_set_port_state

This function sets the port auth_mac state.

Syntax

```
int
hal_auth_mac_set_port_state (unsigned int index, int mode,
                             enum hal_auth_mac_port_state state);
```

Input Parameter

`index`	Port index
`mode`	Port mode
`state`	MAC-based port authentication state

Output Parameter

None

Return Value

< 0 when function fails

HAL_SUCCESS when function succeeds

Quality of Service API

The following subsection includes the Quality of Service (QoS) functions.

Table 2-14: QoS API Functions

Functions	Description
hal_l2_qos_init	This function initializes the QoS hardware layer.
hal_l2_qos_deinit	This function deinitializes the QoS hardware layer.
hal_l2_qos_default_user_priority_set	This function sets the default user priority for a port.
hal_l2_qos_default_user_priority_get	This function gets the default user priority for a port.
hal_l2_qos_regen_user_priority_set	This function sets the regenerated user priority of a port.
hal_l2_qos_regen_user_priority_get	This function gets the regenerated user priority of a port.
hal_l2_qos_traffic_class_set	This function sets the traffic class value for a port.
hal_l2_qos_traffic_class_get	This function gets the traffic class value for a port

hal_l2_qos_init

This function initializes the Quality of Service (QoS) hardware layer.

Syntax

```
int
hal_l2_qos_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_L2_QOS_INIT when function fails

HAL_SUCCESS when function succeeds

hal_l2_qos_deinit

This function deinitializes the QoS hardware layer.

Syntax

```
int
hal_l2_qos_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_L2_QOS_DEINIT

HAL_SUCCESS when function succeeds

hal_l2_qos_default_user_priority_set

This function sets the default user priority for a port.

Syntax

```
int
hal_l2_qos_default_user_priority_set (unsigned int ifindex,
                                      unsigned char user_priority)
```

Input Parameters

`ifindex`	Port interface index
`user_priority`	Default user priority

Output Parameters

None

Return Value

HAL_ERR_L2_QOS

HAL_SUCCESS when function succeeds

hal_l2_qos_default_user_priority_get

This function gets the default user priority for a port.

Syntax

```
int
hal_l2_qos_default_user_priority_get (unsigned int ifindex,
                                      unsigned char *user_priority)
```

Input Parameters

`ifindex`	Port interface index

Output Parameters

`user_priority`	User priority

Return Value

HAL_ERR_L2_QOS

HAL_SUCCESS when function succeeds

hal_l2_qos_regen_user_priority_set

This function sets the regenerated user priority of a port.

Syntax

```
int
hal_l2_qos_regen_user_priority_set (unsigned int ifindex,
                                     unsigned char recvd_user_priority,
                                     unsigned char regen_user_priority);
```

Input Parameters

`ifindex` Port interface index

`recvd_user_priority`

Received user priority

`regen_user_priority`

Regenerated user priority

Output Parameters

None

Return Value

HAL_ERR_L2_QOS

HAL_SUCCESS when function succeeds

hal_l2_qos_regen_user_priority_get

This function gets the regenerated user priority for a port.

Syntax

```
int
hal_l2_qos_regen_user_priority_get (unsigned int ifindex,
                                     unsigned char *regen_user_priority);
```

Input Parameters

`ifindex` Port interface index

Output Parameters

`regen_user_priority`

Regenerated user priority

Return Value

HAL_ERR_L2_QOS

HAL_SUCCESS when function succeeds

hal_l2_qos_traffic_class_set

This function sets the traffic class value for a port for a user priority and traffic class.

Syntax

```
int
hal_l2_qos_traffic_class_set (unsigned int ifindex,
                              unsigned char user_priority,
                              unsigned char traffic_class,
                              unsigned char traffic_class_value)
```

Input Parameters

`ifindex`	Port interface index
`user_priority`	User priority
`traffic_class`	Traffic class
`traffic_class_value`	Traffic class value

Output Parameters

None

Return Value

HAL_ERR_L2_QOS_TRAFFIC_CLASS

HAL_SUCCESS when function succeeds

hal_l2_qos_traffic_class_get

This function gets the traffic class value for a port for a user priority and traffic class.

Syntax

```
int
hal_l2_qos_traffic_class_get (unsigned int ifindex,
                              unsigned char user_priority,
                              unsigned char traffic_class,
                              unsigned char traffic_class_value)
```

Input Parameters

`ifindex`	Port interface index
`user_priority`	User priority
`traffic_class`	Traffic class

Output Parameters

`traffic_class_value`

Traffic class value

Return Value

HAL_ERR_L2_QOS_TRAFFIC_CLASS

HAL_SUCCESS when function succeeds

Port Mirroring API

The following subsection includes the Port Mirroring functions.

Table 2-15: Port Mirroring API Functions

Functions	Description
hal_port_mirror_init	This function initializes the port-mirroring hardware layer.
hal_port_mirror_deinit	This function deinitializes the port-mirroring hardware layer.
hal_port_mirror_set	This function sets the port mirroring.
hal_port_mirror_unset	This function unsets the port mirroring.

hal_port_mirror_init

This function initializes the port-mirroring hardware layer.

Syntax

```
int
hal_port_mirror_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_PMIRROR_INIT when function fails

HAL_SUCCESS when function succeeds

hal_port_mirror_deinit

This function deinitializes the port-mirroring hardware layer.

Syntax

```
int
hal_port_mirror_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_PMIRROR_DEINIT

HAL_SUCCESS when function succeeds

hal_port_mirror_set

This function sets the port mirroring.

Syntax

```
int
hal_port_mirror_set (unsigned int to_ifindex, unsigned int from_ifindex,
                     enum hal_port_mirror_direction direction);
```

Input Parameters

`to_ifindex`	Mirrored to the port
`from_ifindex`	Mirrored from the port
`direction`	Direction to set for mirroring

Output Parameters

None

Return Value

HAL_ERR_PMIRROR_SET

HAL_SUCCESS when function succeeds

hal_port_mirror_unset

This function unsets port mirroring.

Syntax

```
int
hal_port_mirror_unset (unsigned int to_ifindex, unsigned int from_ifindex,
                       enum hal_port_mirror_direction direction);
```

Input Parameters

`to_ifindex`	Mirrored to the port
`from_ifindex`	Mirrored from the port
`direction`	Direction to unset for mirroring

Output Parameters

None

Return Value

HAL_ERR_PMIRROR_UNSET

HAL_SUCCESS when function succeeds

Rate Limit API

The following subsection includes the rate limit functions.

Table 2-16: Rate Limit API Functions

Functions	Description
hal_ratelimit_init	This function initializes rate limiting.
hal_ratelimit_deinit	This function deinitializes rate limiting.
hal_l2_ratelimit_bcast	This function sets the percentage of the port bandwidth.
hal_l2_ratelimit_mcast	This function sets the percentage of the port bandwidth.
hal_l2_ratelimit_dlf_bcast	This function sets the level as a percentage of the port bandwidth for DLF.

hal_ratelimit_init

This function initializes rate limiting.

Syntax

```
int
hal_ratelimit_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_RATELIMIT_INIT when function fails

HAL_SUCCESS when function succeeds

hal_ratelimit_deinit

This function deinitializes rate limiting.

Syntax

```
int
hal_ratelimit_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_RATELIMIT_DEINIT

HAL_SUCCESS when function succeeds

hal_l2_ratelimit_bcast

This function sets the percentage of the port bandwidth devoted to broadcast storm suppression.

Syntax

```
int
hal_l2_ratelimit_bcast (unsigned int ifindex,
                        unsigned char level,
                        unsigned char fraction);
```

Input Parameters

`ifindex`	Port interface index
`level`	Level as a percentage of the port bandwidth
`fraction`	Fraction level as a percentage of the port bandwidth

Output Parameters

None

Return Value

HAL_ERR_RATELIMIT_BCAST

HAL_SUCCESS when function succeeds

hal_l2_ratelimit_mcast

This function sets the percentage of the port bandwidth devoted to multicast.

Syntax

```
int
hal_l2_ratelimit_mcast (unsigned int ifindex,
                        unsigned char level,
                        unsigned char fraction);
```

Input Parameters

`ifindex`	Port interface index
`level`	Level as a percentage of the port bandwidth
`fraction`	Fraction level as a percentage of the port bandwidth

Output Parameters

None

Return Value

HAL_ERR_RATELIMIT_MCAST

HAL_SUCCESS when function succeeds

hal_l2_ratelimit_dlf_bcast

This function sets the level as a percentage of the port bandwidth for DLF (Destination Lookup Failure) broadcast.

Syntax

```
int
hal_l2_ratelimit_dlf_bcast (unsigned int ifindex,
                            unsigned char level,
                            unsigned char fraction);
```

Input Parameters

`ifindex`	Port interface index
`level`	Level as a percentage of the port bandwidth
`fraction`	Fraction level in percentage of the port bandwidth

Output Parameters

None

Return Value

HAL_ERR_RATELIMIT_MCAST

HAL_SUCCESS when function succeeds

VLAN API

The following subsection includes the VLAN functions.

Table 2-17: VLAN API Functions

Functions	Description
hal_vlan_init	This function initializes the VLAN hardware layer component.
hal_vlan_deinit	This function deinitializes the VLAN hardware layer component.
hal_vlan_add	This function adds a VLAN.
hal_vlan_delete	This function deletes a VLAN.
hal_vlan_set_port_type	This function sets an acceptable frame type for a port.
hal_vlan_set_default_pvid	This function sets the default port VLAN ID (PVID).
hal_vlan_add_vid_to_port	This function adds a VLAN to a port.
hal_vlan_delete_vid_from_port	This function deletes a VLAN from a port.

Table 2-17: VLAN API Functions

Functions	Description
hal_vlan_port_set_dot1q_state	This function sets the dot1q state on a port.
hal_vlan_add_cvid_to_port	This function adds a CVLAN to a port.
hal_vlan_delete_cvid_to_port	This function delete a CVLAN to a port.
hal_vlan_create_cvlan	This function creates a mapping between C-VLAN to S-VLAN.
hal_vlan_delete_cvlan	This function deletes a mapping between C-VLAN to S-VLAN.
hal_vlan_create_cvlan_registration_entry	This function creates a mapping between C-VLAN to S-VLAN on a CE port.
hal_vlan_delete_cvlan_registration_entry	This function deletes a mapping between C-VLAN to S-VLAN from a CE port.
hal_vlan_create_vlan_trans_entry	This function creates a translation from VLAN 1 to VLAN 2 on a port.
hal_vlan_delete_vlan_trans_entry	This function deletes a translation from VLAN 1 to VLAN 2 on a port.
hal_vlan_set_native_vid	This function configures the native VLAN for the trunk port.
hal_vlan_set_pro_edge_pvid	This function configures the primary VID (PVID) for the provider edge port.
hal_vlan_set_pro_edge_untagged_vid	This function configures the untagged VID for the egress for the PE port.
hal_vlan_add_pro_edge_port	This function configures the primary VID for the provider edge port.
hal_pro_vlan_set_dtag_mode	This function configures double-tag mode.
hal_vlan_classifier_init	This function initializes the VLAN classifier hardware layer.
hal_vlan_classifier_deinit	This function deinitializes the VLAN classifier hardware layer.
hal_vlan_classifier_add	This function adds a VLAN classification group.
hal_vlan_classifier_del	This function deletes a VLAN classification group.
hal_vlan_stacking_enable	This function enables VLAN Stacking on an interface.
hal_vlan_stacking_disable	This function disables VLAN Stacking on an interface.

hal_vlan_init

This function initializes the VLAN hardware layer component.

Syntax

```
int
hal_vlan_init (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_VLAN_INIT when function fails

HAL_SUCCESS when function succeeds

hal_vlan_deinit

This function deinitializes the VLAN hardware layer component.

Syntax

```
int hal_vlan_deinit (void);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_VLAN_DEINIT

HAL_SUCCESS when function succeeds

hal_vlan_add

This function adds a VLAN.

Syntax

```
int
hal_vlan_add (char *name, enum hal_vlan_type type, unsigned short vid);
```

Input Parameters

`name`	Bridge name
`type`	VLAN type
`vid`	VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete

This function deletes a VLAN.

Syntax

```
int
hal_vlan_delete (char *name, enum hal_vlan_type type, unsigned short vid);
```

Input Parameters

`name`	Bridge name
`type`	VLAN type
`vid`	VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_set_port_type

This function sets an acceptable frame type for a port.

Syntax

```
int
hal_vlan_set_port_type (char *name,
                        unsigned int ifindex,
                        enum hal_vlan_port_type port_type,
                        enum hal_vlan_port_type sub_port_type,
                        enum hal_vlan_acceptable_frame_type acceptable_frame_type,
                        unsigned short enable_ingress_filter)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`port_type`	Trunk, access, or hybrid
`sub_port_type`	Sub port type
`acceptable_frame_type`	Valid frame type
`enable_ingress_filter`	Enable ingress filtering

Output Parameters

None

Return Value

HAL_ERR_VLAN_FRAME_TYPE

HAL_SUCCESS when function succeeds

hal_vlan_set_default_pvid

This function sets the default port VLAN ID (PVID).

Syntax

```
int
hal_vlan_set_default_pvid (char *name, unsigned int ifindex,
                           unsigned short pvid,
                           enum hal_vlan_egress_type egress)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`pvid`	Default PVID
`egress`	Egress tagged/untagged

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_add_vid_to_port

This function adds a VLAN to a port.

Syntax

```
int
hal_vlan_add_vid_to_port (char *name, unsigned int ifindex,
                          unsigned short vid,
                          enum hal_vlan_egress_type egress)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`vid`	VLAN ID
`egress`	Egress tagged/untagged

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete_vid_from_port

This function deletes a VLAN from a port.

Syntax

```
int
hal_vlan_delete_vid_from_port (char *name, unsigned int ifindex,
                               unsigned short vid)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`vid`	VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_port_set_dot1q_state

This function sets the dot1q state on a port.

Syntax

```
int
hal_vlan_port_set_dot1q_state (unsigned int ifindex, unsigned short enable,
                               unsigned short enable_ingress_filter);
```

Input Parameters

`ifindex`	Interface index
`enable`	To enable or disable dot1q
`enable_ingress_filter`	Enable ingress filtering

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_add_cvid_to_port

This function adds a CVLAN to a port.

Syntax

```
int
hal_vlan_add_cvid_to_port (char *name, unsigned int ifindex,
                           unsigned short cvid,
                           unsigned short svid,
                           enum hal_vlan_egress_type egress);
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`cvid`	C-VLAN ID
`svid`	S-VLAN ID
`egress`	Egress tagged/untagged

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete_cvid_to_port

This function delete a CVLAN to a port.

Syntax

```
int
hal_vlan_delete_cvid_from_port (char *name, unsigned int ifindex,
                                unsigned short cvid,
                                unsigned short svid);
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`cvid`	C-VLAN ID
`svid`	S-VLAN ID
`egress`	Egress tagged/untagged

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_create_cvlan

This function creates a mapping between C-VLAN to S-VLAN and creates a corresponding Internal VLAN.

Syntax

```
int
hal_vlan_create_cvlan (char *name, unsigned short cvid,
                       unsigned short svid);
```

Input Parameters

name	Bridge name
cvid	C-VLAN ID
svid	S-VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete_cvlan

This function deletes a mapping between C-VLAN to S-VLAN.

Syntax

```
int
hal_vlan_delete_cvlan (char *name, unsigned short cvid,
                       unsigned short svid);
```

Input Parameters

name	Bridge name
cvid	C-VLAN ID
svid	S-VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_create_cvlan_registration_entry

This function creates a mapping between C-VLAN to S-VLAN on a CE port.

Syntax

```
int
hal_vlan_create_cvlan_registration_entry (char *name, unsigned int ifindex,
                                          unsigned short cvid,
                                          unsigned short svid);
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete_cvlan_registration_entry

This function deletes a mapping between C-VLAN to S-VLAN on a CE port.

Syntax

```
int
hal_vlan_delete_cvlan_registration_entry (char *name, unsigned int ifindex,
                                          unsigned short cvid,
                                          unsigned short svid);
```

Input Parameters

name	Bridge name
ifindex	Interface index
cvid	C-VLAN ID
svid	S-VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_create_vlan_trans_entry

This function creates a translation from VLAN 1 to VLAN 2 on a port.

Syntax

```
int
hal_vlan_create_vlan_trans_entry (char *name, unsigned int ifindex,
                                   unsigned short vid,
                                   unsigned short trans_vid);
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`vid`	VLAN ID to be translated
`trans_vid`	Translated VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_delete_vlan_trans_entry

This function deletes a translation from VLAN1 to VLAN 2 on a CN port.

Syntax

```
int
hal_vlan_delete_vlan_trans_entry (char *name, unsigned int ifindex,
                                  unsigned short vid,
                                  unsigned short trans_vid);
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index

`vid`	VLAN ID to be translated
`trans_vid`	Translated VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_NOT_EXISTS

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_set_native_vid

This function configures the native VLAN for the trunk port.

Syntax

```
int
hal_vlan_set_native_vid (char *name, unsigned int ifindex,
                         unsigned short vid)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`native_vid`	Native VLAN ID

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_set_pro_edge_pvid

This function configures the primary VID (PVID) for the provider edge port.

Syntax

```
int
hal_vlan_set_pro_edge_pvid (char *name, unsigned int ifindex,
                            unsigned short svid,
                            unsigned short pvid);
```

Input Parameters

`name`	Bridge name

`ifindex`	Interface index
`svid`	VLAN ID of the Provider Edge Port
`pvid`	VLAN ID used for Untagged Packets

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_set_pro_edge_untagged_vid

This function configures the untagged VID for the egress for the provider edge port.

Syntax

```
int
hal_vlan_set_pro_edge_untagged_vid (char *name, unsigned int ifindex,
                                     unsigned short svid,
                                     unsigned short untagged_vid)
```

Input Parameters

`name`	Bridge name
`ifindex`	Interface index
`svid`	VLAN ID of the Provider Edge Port
`untagged_vid`	VLAN ID that is transmitted untagged in the provider Edge Port.

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_add_pro_edge_port

This function configures the primary VID for the provider edge port.

Syntax

```
int
hal_vlan_add_pro_edge_port (char *name, unsigned int ifindex,
                             unsigned short svid);
```

Input Parameters

name	Bridge name
ifindex	Interface index
svid	VLAN ID of the Provider Edge Port

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_del_pro_edge_port

This function configures the Untagged VID for the Egress for the Provider Edge Port.

Syntax

```
int
hal_vlan_del_pro_edge_port (char *name, unsigned int ifindex,
                            unsigned short svid)
```

Input Parameters

name	bridge name
ifindex	Interface index
svid	VLAN id of the Provider Edge Port

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_pro_vlan_set_dtag_mode

This function configures double-tag mode.

Syntax

```
int
hal_pro_vlan_set_dtag_mode (unsigned int ifindex,
                            unsigned short dtag_mode)
```

Input Parameters

`ifindex`	Interface index
`dtag_mode`	Whether it is a single tag port or a double tag port

Output Parameters

None

Return Value

HAL_ERR_VLAN_PORT_NOT_EXISTS

HAL_ERR_VLAN_NOT_EXISTS

HAL_SUCCESS when function succeeds

hal_vlan_classifier_init

This function initializes the VLAN classifier hardware layer.

Syntax

```
int
hal_vlan_classifier_init();
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_VLAN_CLASSIFIER_INIT when function fails

HAL_SUCCESS when function succeeds

hal_vlan_classifier_deinit

This function deinitializes the VLAN classifier hardware layer.

Syntax

```
int hal_vlan_classifier_deinit();
```

Input Parameters

None

Output Parameters

None

Return Value

HAL_ERR_VLAN_CLASSIFIER_DEINIT

HAL_SUCCESS when function succeeds

hal_vlan_classifier_add

This function adds a VLAN classification group.

Syntax

```
int
hal_vlan_classifier_add (struct hal_vlan_classifier_rule *rule_ptr,u_int32_t
                        ifindex, u_int32_t refcount)
```

Input Parameters

`rule_ptr`	VLAN classification rule
`ifindex`	Interface index
`refcount`	Reference count

Output Parameters

None

Return Value

HAL_ERR_VLAN_CLASSIFIER_ADD

HAL_SUCCESS when function succeeds

hal_vlan_classifier_del

This function deletes a VLAN classification group.

Syntax

```
int
hal_vlan_classifier_del (struct hal_vlan_classifier_rule *rule_ptr,u_int32_t ifindex,
                        u_int32_t refcount)
```

Input Parameters

`rule_ptr`	VLAN classification rule
`ifindex`	Interface index
`refcount`	Reference count

Output Parameters

None

Return Value

HAL_ERR_VLAN_CLASSIFIER_ADD

HAL_SUCCESS when function succeeds

hal_vlan_stacking_enable

This function enables VLAN Stacking on an interface.

Syntax

```
int
hal_vlan_stacking_enable (u_int32_t ifindex,
                          u_int16_t ethtype,
                          u_int16_t stackmode)
```

Input Parameters

`ifindex`	Interface index
`ethtype`	Ethernet type value for the VLAN tag
`stackmode`	VLAN stacking mode

Output Parameters

None

Return Value

HAL_SUCCESS when function succeeds on success

< 0 when function fails

hal_vlan_stacking_disable

This function disables VLAN Stacking on an interface.

Syntax

```
int
hal_vlan_stacking_disable (u_int32_t ifindex,
                           u_int16_t ethtype,
                           u_int16_t stackmode)
```

Input Parameters

`ifindex`	Interface index
`ethtype`	Ethernet type value for the VLAN tag
`stackmode`	VLAN stacking mode

Output Parameters

None

Return Value

HAL_SUCCESS when function succeeds

< 0 when function fails

CHAPTER 3 Hardware Services Layer

The Hardware Services Layer (HSL) is a socket back-end layer that implements the hardware system-related functionality. This chapter includes an overview of HSL, lists the data structures, defines the API callbacks, and describes API functions.

Overview

A socket interface receives configuration updates and then sends responses and notifications. Message encapsulation and message send are similar to HAL; however, HSL does not require a response or acknowledgement message. Moreover, HSL acknowledges or sends a response to all messages.

HSL Components

The following subsection describes the Hardware Service Layer (HSL) components:

Interface Management

The Interface Manager is the master interface manager for the system. It provides the following functionality:

- A logical view of the interfaces in the system to the control plane interface manager
- Manages and configures the system interfaces that are not managed by the network processor or ASIC
- Manages the configuration and management of the hardware interfaces on the network processor or ASIC
- For layer 3 interfaces, populates the interface in the TCP/IP stack for slow-path and exception packet handling
- Provides an abstracted TCP/IP interface that can populate by different TCP/IP stacks with the same OS
- Provides an abstracted hardware interface
- Provides maintenance of the database and hierarchies for all interface types and their relationships, including API functions that maintain and change these hierarchies through the operation of the control plane
- Includes a registration callback mechanism

FIB Manager

RIB (Routing Information Database) and FIB (Forwarding Information Database) store forwarding data to be used by routers. RIB collects all forwarding possibilities known to a router that was learned from different sources. FIB includes an active selection of forwarding entries used by a router to forward data. The FIB manager stores and programs all of these forwarding entries that are selected for forwarding to the network or ASIC and operating system.

The two paths for traffic forwarding include fast path and slow path. With fast path, traffic moves through pre-programmed entries on the network processor or ASIC. With slow path, the network processor or ASIC cannot forward the traffic, so the data is sent to the CPU/OS first before being processed. The FIB manager then removes forwarding entries from the network processor or ASIC and OS when entries are withdrawn from the FIB.

Typically, both the OS and chip are identically programmed, and only control traffic goes to the CPU. In addition, there is a memory limitation on processors for the number of prefixes a chip can accommodate. Thus, modules keep a shadow of installed forwarding entries in both the chip and OS. The FIB manager database has extra fields specific to the hardware, since the processor and ASIC programming requires extra information.

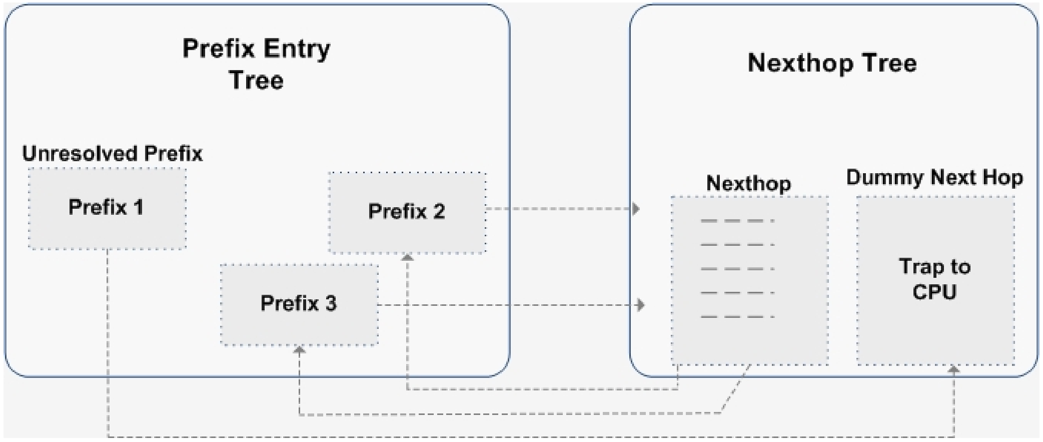

Figure 3-1: FIB Manager

Packet Driver

The packet driver module processes packets. Packet receipt occurs in the hardware-interrupt context, when the packet descriptor is passed to the user-callback function. To avoid a packet drop in case of bulk, the packet driver does not immediately start parsing and processing packets. Instead, the packet driver implements a first-in, first out (FIFO) queue for packets to be processed. During hardware interrupt, the packet descriptor is stored in the queue, and the packet-ready semaphore is released. If the queue is full, or the system is not ready for traffic, packets are dropped. Packet processing is based on the hardware CPU error code.

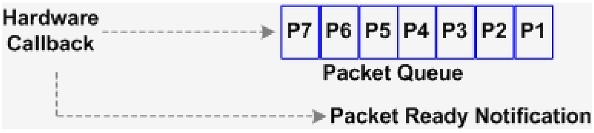

Figure 3-2: Packet Driver Flow

Ethernet Driver

The Ethernet driver module handles interfaces in the OS, passes traffic between the hardware and the OS stack, and manages network buffer pools.

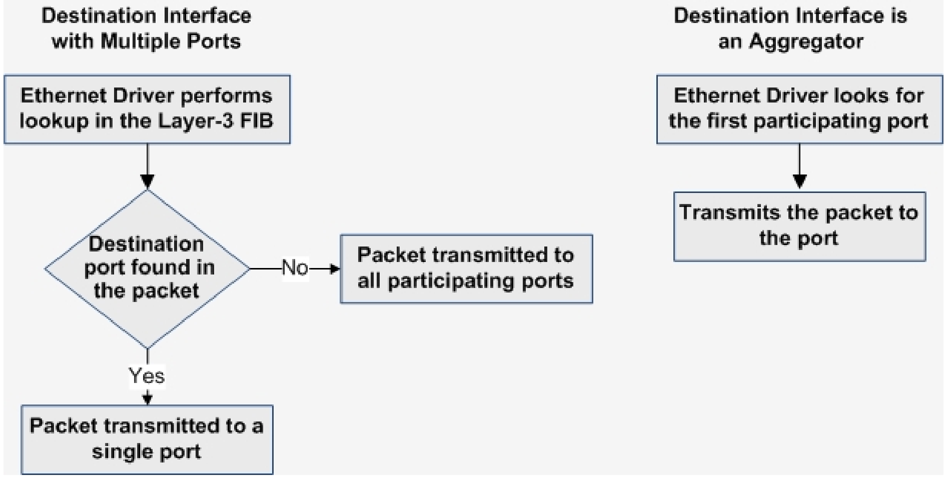
Destination Interface with Multiple Ports
Ethernet Driver performs lookup in the Layer-3 FIB
Destination port found in the packet
No
Packet transmitted to all participating ports
Yes
Packet transmitted to a single port
Destination Interface is an Aggregator
Ethernet Driver looks for the first participating port
Transmits the packet to the port

Figure 3-3: Ethernet Driver Sequence

HSL Interfaces

The following are the related system interfaces supported by HSL.

Socket Interface to Control Plane

The socket interface receives control plane messages (for example, configuration messages) and sends the response and notification.

SDK Interface to Hardware

The SDK (software development kit) interface is used to program the hardware chip set (fast path) according to control plane messages. It is also retrieves counters and statistics.

OS Interface

The OS interface is used by the operating system for slow path forwarding configuration according to the control plane messages. It also maintains a full forwarding database, which is not restricted by hardware limitations.

Event Interface

The event interface registers the number of callbacks with the hardware and OS to receive and process system events. It is used for hardware module attachment/detachment, link/auto-negotiation status update, and new address addition/notification from the OS. HSL processes events and notifies the control modules and system modules of the change. For example, the detaching of an interface requires a notification to the control plane to stop all the protocols running on the detached interface. It also requires a notification to the interface manager to delete the detached interface and any associated configuration.

Data Structures

The following subsection list the data structures for HSL.

hsl_bridge

This data structure helps manage bridge functions. It is defined in libhsl/L2/hsl_bridge.h:

Type	Definition
name[HAL_BRIDGE_NAME_LEN + 1];	Character name
ageing_time	Ageing time.
flags	Flags, including: HSL_BRIDGE_LEARNING (1 << 0) HSL_BRIDGE_VLAN_AWARE (1 << 1) HSL_STP_INSTANCE_MAX (8) HSL_MSTP_CIST_INSTANCE (0) HSL_BRIDGE_GVRP_ENABLED 1
hal_bridge_type type	enum ;
hsl_avl_tree *vlan_tree	Map of VLAN ports that are of type struct hsl_vlan_port.
hsl_avl_tree *port_tree	Tree of ports.
*system_info	Platform specific information.

Type	Definition
hsl_vid_entry *inst_table[HSL_MAX_MSTP_INSTANCES];	Mapping of VID to instances
hsl_inst_info inst_info_table[HSL_MAX_MSTP_INSTANCES];	Mapping of VID to instances

Defintion

```
struct hsl_bridge
{
  char name[HAL_BRIDGE_NAME_LEN + 1]; /* Name. */
  u_int32_t ageing_time;              /* Ageing time. */
  u_char flags;                       /* Flags. */
#define HSL_BRIDGE_LEARNING          (1 << 0)
#define HSL_BRIDGE_VLAN_AWARE        (1 << 1)
#define HSL_STP_INSTANCE_MAX         (8)
#define HSL_MSTP_CIST_INSTANCE       (0)

#define HSL_BRIDGE_GVRP_ENABLED     1
  enum hal_bridge_type type;
#ifdef HAVE_VLAN
  struct hsl_avl_tree  *vlan_tree;           /* Map of VLAN->ports.
                                                of type 'struct hsl_vlan_port'. */
#endif /* HAVE_VLAN. */
  u_char gxrp_enable;
  struct hsl_avl_tree *port_tree;            /* Tree of ports. */
  void *system_info;                         /* Platform specific information. */
#ifdef HAVE_PROVIDER_BRIDGE
  char edge;
#endif /* HAVE_PROVIDER_BRIDGE */
  /* Mapping of VID to instances  */
  struct hsl_vid_entry *inst_table[HSL_MAX_MSTP_INSTANCES];
  struct hsl_inst_info inst_info_table[HSL_MAX_MSTP_INSTANCES];
};
```

hsl_bridge_port

This data structure helps manage bridge port functions. It is defined in libhsl/L2/hsl_bridge.h:

Type	Definition
hsl_if *ifp	Back pointer to ifp.
hsl_bridge *bridge	Back pointer to bridge.
hal_vlan_port_type type	VLAN port type, including access, trunk, and hybrid.
hal_vlan_port_type sub_type	VLAN port sub-type, including access, trunk, and hybrid.
hsl_port_vlan *vlan	VLAN information.

Definition

```
struct hsl_bridge_port
{
  struct hsl_if *ifp;                        /* Backpointer to ifp. */
  struct hsl_bridge *bridge;                 /* Backpointer to bridge. */
  enum hal_vlan_port_type type;              /* Access, trunk, hybrid. */
  enum hal_vlan_port_type sub_type;          /* Access, trunk, hybrid. */

#ifdef HAVE_L2LERN
  struct hsl_mac_access_grp *hsl_macc_grp;
  u_int32_t stp_port_state;
#endif /* HAVE_L2LERN */
#ifdef HAVE_VLAN
  struct hsl_port_vlan *vlan;                /* VLAN information. */
#endif /* HAVE_VLAN. */
#ifdef HAVE_PVLAN
  int pvlan_port_mode;
  void *system_info;
#endif /* HAVE_PVLAN */
#ifdef HAVE_MAC_AUTH
  int auth_mac_port_ctrl;
#define AUTH_MAC_ENABLE   (1<<0)
#endif /* HAVE_MAC_AUTH */
#ifdef HAVE_PROVIDER_BRIDGE
  struct hsl_avl_tree *reg_tab;
  struct hsl_bridge_protocol_process proto_process;
#endif /* HAVE_PROVIDER_BRIDGE */
};
```

hsl_bridge_master

This data structure helps manage bridge master functions. It is defined in libhsl/L2/hsl_bridge.h:

Type	Definition
hsl_bridge *bridge	Currently, only one bridge is supported.
hsl_mac_address_t lldp_addr;	Configurable LLDP Multicast MAC Address.
ipi_sem_id mutex	Mutex
hsl_mac_address_t dest_addr [HAL_PROTO_MAX]	Configurable protocol multicast MAC address
cfm_ether_type	Configurable CFM Ethernet type.
cfm_cc_levels	CFM levels that are enabled
cfm_tr_levels	CFM levels that are enabled
hsl_l2_hw_callbacks *hw_cb	Hardware callbacks

Definition

```
struct hsl_bridge_master
{
  struct hsl_bridge *bridge;               /* Currently only 1 bridge supported. */
  hsl_mac_address_t lldp_addr;            /* Configurable LLDP Multicast MAC Address */
  ipi_sem_id mutex;                        /* Mutex. */
  /* Configurable Protocol Multicast MAC Address */
  hsl_mac_address_t dest_addr [HAL_PROTO_MAX];
#ifdef HAVE_ONMD
  u_int16_t cfm_ether_type;     /* Configurable CFM Ethernet Type */
  u_int8_t cfm_cc_levels;       /* CFM Levels that are enabled */
  u_int8_t cfm_tr_levels;       /* CFM Levels that are enabled */
#endif /* HAVE_ONMD */
  struct hsl_l2_hw_callbacks *hw_cb;         /* Hardware callbacks. */
};
```

hsl_vlan_port

This data structure helps manage VLAN port functions. It is defined in libhsl/L2/hsl_vlan.h:

Type	Definition
hsl_vid_t vid	VLAN identifier
hsl_avl_tree *port_tree;	Tree of ports on which VLAN exists
hsl_avl_tree *vlan_tre	Tree of secondary VLANs attached to the primary VLAN.

Definition

```
struct hsl_vlan_port
{
  hsl_vid_t      vid;                   /* VLAN id. */
  struct hsl_avl_tree *port_tree;       /* Tree of ports on which VLAN exists. */
#ifdef HAVE_PVLAN
  enum hal_pvlan_type vlan_type;
  struct hsl_avl_tree *vlan_tree;       /* Tree of secondary vlans attached
                                           to primary vlan */
  void *system_info;
#endif /* HAVE_PVLAN */
#ifdef HAVE_L2LERN
  struct hsl_vlan_access_map *hsl_vacc_map;
#endif /* HAVE_L2LERN */
};
```

hsl_vlan_port_attr

This data structure helps manage FDB (forwarding database) entry functions.is defined in libhsl/L2/hsl_vlan.h:

Type	Definition
vid	VLAN identifier
etagged	Egress tagged

Definition

```
    struct hsl_vlan_port_attr
    {
       hsl_vid_t      vid;                    -- VLAN id.
       HSL_BOOL       etagged;                -- Egress traffic is tagged.
```

```
   };
```

hsl_port_vlan

This data structure helps manage port VLAN functions. It is defined in libhsl/L2/hsl_vlan.h:

Type	Definition
*port	Back pointer to the bridge port
mode	Port mode
pvid	Access port, including access, Trunk port: native
flags	Flags, including: NSM_VLAN_ENABLE_INGRESS_FILTER (1 << 0) NSM_VLAN_ACCEPTABLE_FRAME_TYPE_TAGGED (1 << 1)
*vlan_tree	VLANs this port belongs to

Definition

```
struct hsl_port_vlan
{
  struct hsl_bridge_port    *port;     /* Backpointer to bridge port. */
  u_char  mode;                 /* Port mode. */

  hsl_vid_t                 pvid;      /* Access port: access, Trunk port: native. */

  u_char flags;                        /* Flags. */
#define NSM_VLAN_ENABLE_INGRESS_FILTER        (1 << 0)
#define NSM_VLAN_ACCEPTABLE_FRAME_TYPE_TAGGED (1 << 1)
  struct hsl_avl_tree *vlan_tree;     /* VLANs this port belongs to. */
#ifdef HAVE_PVLAN
  u_char pvlan_port_mode;
#endif /* HAVE_PVLAN */
};
```

hsl_if_resv_vlan

This data structure helps manage interface reservation VLANs. It is defined in libhsl/common/hsl_ifmgr.h:

Type	Definition
Interface types typedef enum	Interface types, including: HSL_IF_TYPE_UNK = 0, /* Unknown. */ HSL_IF_TYPE_LOOPBACK = 1, /* Loopback. */ HSL_IF_TYPE_IP = 2, /* IP. */ HSL_IF_TYPE_L2_ETHERNET = 3, /* Ethernet. */ HSL_IF_TYPE_MPLS = 4, /* MPLS. */ HSL_IF_TYPE_TUNNEL = 5 /* Tunnel */

IP forwarding mode typedef enum	IP forwarding mode, including:
	HSL_IF_IP_FORWARDING_ENABLE = 1, Enable IP forwarding.
	HSL_IF_IP_FORWARDING_DISABLE = 2 Disable IP forwarding.

Type	Definition

Operational Status code
typedef enum
	Operational Status code, including:
	HSL_IF_OPER_STATUS_UP = 1, /* Operationally UP */
	HSL_IF_OPER_STATUS_DOWN = 2, /* Operationally DOWN */
	HSL_IF_OPER_STATUS_UNKNOWN = 3 /* Status unknown */

Switching type for a interface
typedef enum
	Switching type for a interface, including:
	HSL_IF_SWITCH_L2 = 1, /* L2 switching only. */
	HSL_IF_SWITCH_L2_L3 = 2, /* L2/L3 switching. */
	HSL_IF_SWITCH_L3 = 3 /* L3 switching only. */

typedef enum
Administrative Status code
	Administrative Status code, including:
	HSL_IF_ADMIN_STATUS_UP = 1, /* Administratively UP */
	HSL_IF_ADMIN_STATUS_DOWN = 2 /* Administratively DOWN */

typedef struct _hsl_prefix_list	IP interface address prefix list.
hsl_prefix_t prefix;	Prefix
struct _hsl_prefix_list *next;	Next
flags	Flags
struct _hsl_ip_if	IPv4 Interface.
mtu	IPv(4|6) MTU
nucAddr	Number of UC IP address.
*ucAddr;	Array of unicast IP addresses.
mode	IPv(4|6) forwarding mode.

Definition

```
struct hsl_if_resv_vlan;
typedef struct hsl_if_resv_vlan hsl_if_resv_vlan_t;
/*
  Interface properties definition.
 */
#define HSL_IF_CPU_ONLY_INTERFACE                    (0x10)
/*
   Interface types.
*/
typedef enum
  {
    HSL_IF_TYPE_UNK         = 0,            /* Unknown. */
    HSL_IF_TYPE_LOOPBACK    = 1,            /* Loopback. */
    HSL_IF_TYPE_IP          = 2,            /* IP. */
    HSL_IF_TYPE_L2_ETHERNET = 3,            /* Ethernet. */
    HSL_IF_TYPE_MPLS        = 4,            /* MPLS. */
    HSL_IF_TYPE_TUNNEL      = 5             /* Tunnel */
  } hsl_ifType_t;
/*
   IP forwarding mode.
*/
typedef enum
  {
    HSL_IF_IP_FORWARDING_ENABLE = 1,    /* Enable IP forwarding. */
    HSL_IF_IP_FORWARDING_DISABLE = 2    /* Disable IP forwarding. */
  } hsl_IfMode_t;
```

```
/*
    Operational Status code.
*/
typedef enum
  {
    HSL_IF_OPER_STATUS_UP = 1,          /* Operationally UP   */
    HSL_IF_OPER_STATUS_DOWN = 2,        /* Operationally DOWN */
    HSL_IF_OPER_STATUS_UNKNOWN = 3      /* Status unknown     */
  } hsl_IfOperStatus_t;
/*
  Switching type for a interface
*/
typedef enum
  {
    HSL_IF_SWITCH_L2 = 1,               /* L2 switching only. */
    HSL_IF_SWITCH_L2_L3 = 2,            /* L2/L3 switching. */
    HSL_IF_SWITCH_L3 = 3                /* L3 switching only. */
  } hsl_IfSwitchType_t;
/*
   Administrative Status code.
*/
typedef enum
  HSL_IF_ADMIN_STATUS_UP = 1,           /* Administratively UP   */
    HSL_IF_ADMIN_STATUS_DOWN = 2          /* Administratively DOWN */
  } hsl_IfAdminStatus_t;
/*
   IP interface address prefix list.
*/
typedef struct _hsl_prefix_list
{
  /* Prefix. */
  hsl_prefix_t prefix;
  /* Next. */
  struct _hsl_prefix_list *next;
  /* Flags. */
  u_char flags;
#define HSL_IFMGR_IP_ADDR_SECONDARY     (1 << 0)
} hsl_prefix_list_t;
/*
   IPv4 Interface.
*/
struct _hsl_ip_if
{
  u_int16_t mtu;                   /* IPv(4|6) MTU. */
  u_int16_t nucAddr;               /* Number of UC IP address. */
  hsl_prefix_list_t *ucAddr;       /* Array of unicast IP addresses. */
  hsl_IfMode_t mode;               /* IPv(4|6) forwarding mode. */
};
```

hsl_ifmgr_os_callbacks

This data structure helps manage interface manager OS callbacks. It is defined in libhsl/common/hsl_if_os.h:

Type	Definition
int (*os_if_init) (void)	Interface manager OS initialization
int (*os_if_deinit)	Interface manager OS deinitialization
void (*os_if_dump) (struct hsl_if *ifp)	Dump
int (*os_l2_if_flags_set) (struct hsl_if *ifp, unsigned long flags);	Set L2 port flags.
int (*os_l2_if_flags_unset) (struct hsl_if *ifp, unsigned long flags)	Unset L2 port flags
void *(*os_l3_if_configure) (struct hsl_if *ifp, char *name, u_char *hwaddr, int hwaddrlen, hsl_ifIndex_t *ifindex);	Create L3 interface
int (*os_l3_if_unconfigure) (struct hsl_if *ifp);	Delete L3 for an interface
int (*os_l3_if_mtu_set) (struct hsl_if *ifp, int mtu	Set MTU for an interface
int (*os_l3_if_duplex_set) (struct hsl_if *ifp, int duplex)	Set duplex for an interface
int (*os_l3_if_autonego_set) (struct hsl_if *ifp, int autonego);	Set auto-negotiation for an interface
int (*os_l3_if_hwaddr_set) (struct hsl_if *ifp, int hwaddrlen, u_char *hwaddr);	Set hardware address for an interface.

Definition

```
struct hsl_ifmgr_os_callbacks
{
  /* Interface manager OS initialization. */
  int (*os_if_init) (void);
  /* Interface manager OS deinitialization. */
  int (*os_if_deinit) (void);
  /* Dump. */
  void (*os_if_dump) (struct hsl_if *ifp);
  /* Set L2 port flags.
     Parameters:
     IN - interface pointer
     IN -> flags - flags
  */
  int (*os_l2_if_flags_set) (struct hsl_if *ifp, unsigned long flags);
 /* Unset L2 port flags.
     Parameters:
     IN -> interface pointer
     IN -> flags - flags
  */
  int (*os_l2_if_flags_unset) (struct hsl_if *ifp, unsigned long flags);
  /* Create L3 interface.
     Parameters:
     IN -> name - interface name
```

```
    IN -> hwaddr - hardware address
    IN -> hwaddrlen - hardware address length
    OUT -> ifindex - interface index of the OS L3 interface
    Returns:
    OS L3 interface pointer as void *
    NULL on error
 */
 void *(*os_l3_if_configure) (struct hsl_if *ifp, char *name, u_char *hwaddr,
                              int hwaddrlen, hsl_ifIndex_t *ifindex);
 /* Delete L3 interface.
    Parameters:
    IN -> interface pointer
Returns:
    0 on success
    < 0 on error
 */
 int (*os_l3_if_unconfigure) (struct hsl_if *ifp);
 /* Set MTU for interface.
    Parameters:
    IN -> ifp - interface pointer
    IN -> mtu - mtu
    Returns:
    0 on success
    < 0 on error
 */
 int (*os_l3_if_mtu_set) (struct hsl_if *ifp, int mtu);
 /* Set DUPLEX for interface.
    Parameters:
    IN -> ifp - interface pointer
    IN -> duplex - duplex
    Returns:
    0 on success
    < 0 on error
 */
int (*os_l3_if_duplex_set) (struct hsl_if *ifp, int duplex);
 /* Set AUTONEGO for interface.
    Parameters:
    IN -> ifp - interface pointer
    IN -> autonego - autonego
    Returns:
    0 on success
    < 0 on error
 */
 int (*os_l3_if_autonego_set) (struct hsl_if *ifp, int autonego);
 /* Set HW address for a interface.
    Parameters:
    IN -> ifp - interface pointer
    IN -> hwadderlen - address length
    IN -> hwaddr - address
 */
```

```
  int (*os_l3_if_hwaddr_set) (struct hsl_if *ifp, int hwaddrlen, u_char *hwaddr);
;
```

hsl_ifmgr_hw_callbacks

This data structure helps manage interface manager hardware callbacks. It is defined in libhsl/common/hsl_if_hw.h:

Type	Definition
int (*hw_if_init) (void);	Initialization interface manager hardware.
int (*hw_if_deinit) (void)	Deinitialization interface manager hardware.
void (*hw_if_dump) (struct hsl_if *ifp);	Dump
int (*hw_l2_unregister) (struct hsl_if *ifp);	Unregister layer 2 port.
int (*hw_l2_if_flags_set) (struct hsl_if *ifp, unsigned long flags)	Set layer 2 port flags.
int (*hw_l2_if_flags_unset) (struct hsl_if *ifp, unsigned long flags);	Unsets layer 2 port flags
int (*hw_if_packet_types_set) (struct hsl_if *ifp, unsigned long pkt_flags);	Sets the packet type acceptable from this port.
int (*hw_if_packet_types_unset) (struct hsl_if *ifp, unsigned long pkt_flags);	Unsets packet types acceptable from this port.
int (*hw_if_mtu_set) (struct hsl_if *ifp, int mtu);	Set MTU for interface.
int (*hw_if_portbased_vlan) (struct hsl_if *ifp, struct hal_port_map pbitmap);	Add/Remove members for port-based VLAN group
int (*hw_if_cpu_default_vlan) (int vid);	Set CPU port default VLAN identifier
int (*hw_if_wayside_default_vlan) (int vid);	Set wayside port default VLAN identifier
int (*hw_if_preserve_ce_cos)(struct hsl_if *ifp);	Preserve CE COS
int (*hw_if_port_egress) (struct hsl_if *ifp, int egress);	Set port egress mode
int (*hw_if_set_force_vlan) (struct hsl_if *ifp, int vid);	Set force VLAN
int (*hw_if_set_sw_reset) (void);	Sets software reset
int (*hw_if_l3_mtu_set) (struct hsl_if *ifp, int mtu);	Sets MTU for layer 3 interface.
int (*hw_if_duplex_set) (struct hsl_if *ifp, int duplex);	Sets duplex for an interface.
int (*hw_if_autonego_set) (struct hsl_if *ifp, int autonego);	Sets auto-negotiation for an interface
int (*hw_if_bandwidth_set) (struct hsl_if *ifp, u_int32_t bandwidth)	Sets bandwidth for an interface
int (*hw_if_hwaddr_set) (struct hsl_if *ifp, int hwaddrlen, u_char *hwaddr);	Sets a hardware address for an interface.
int (*hw_if_secondary_hwaddrs_set) (struct hsl_if *ifp, int hwaddrlen, int num, u_char **addresses);	Sets secondary hardware addresses for an interface.
int (*hw_if_secondary_hwaddrs_add) (struct hsl_if *ifp, int hwaddrlen, int num, u_char **addresses)	Adds a secondary hardware addresses for an interface.
int (*hw_if_secondary_hwaddrs_delete) (struct hsl_if *ifp, int hwaddrlen, int num, u_char **addresses);	Delete a secondary hardware addresses from an interface.
void *(*hw_l3_if_configure) (struct hsl_if *ifp, void *data);	Creates a layer 3 interface.

Type	Definition
int (*hw_if_post_configure) (struct hsl_if *ifpp, struct hsl_if *ifpc);	Performs any post configuration. This can typically be done after some interface binding is performed.
int (*hw_if_pre_unconfigure) (struct hsl_if *ifpp, struct hsl_if *ifpc);	Perform any pre-unconfiguration. This can typically be done before some interface unbinding is performed.
int (*hw_l3_if_unconfigure) (struct hsl_if *ifp);	Deletes a layer 3 interface.
int (*hw_set_switching_type) (struct hsl_if *ifp, hsl_IfSwitchType_t type);	Sets switching type for a port.
int (*hw_l3_if_flags_set) (struct hsl_if *ifp, unsigned long flags);	Sets layer 3 port flags.
int (*hw_l3_if_flags_unset) (struct hsl_if *ifp, unsigned long flags);	Unsets layer 3 port flags.
int (*hw_l3_if_address_add) (struct hsl_if *ifp, hsl_prefix_t *prefix, u_char flags);	Add an IP address to an interface.
int (*hw_l3_if_address_delete) (struct hsl_if *ifp, hsl_prefix_t *prefix);	Deletes an IP address from an interface.
int (*hw_if_get_counters) (struct hsl_if *ifp);	Gets an interface MAC counter.
int (*hw_if_clear_counters) (struct hsl_if *ifp);	Clear the interface counter
int (*hw_if_mdix_set) (struct hsl_if *ifp, int mdix);	MDIX crossover.
int (*hw_l3_if_bind_fib) (struct hsl_if *ifp, hsl_fib_id_t fib_id);	Binds an interface for a FIB
int (*hw_l3_if_unbind_fib) (struct hsl_if *ifp, hsl_fib_id_t fib_id);	Unbinds an interface from a FIB
int (*hw_if_init_portmirror) (void);	Deinitializes port mirroring.
int (*hw_if_deinit_portmirror) (void)	Sets port mirroring
int (*hw_if_set_portmirror) (struct hsl_if *ifp, struct hsl_if *ifp2, enum hal_port_mirror_direction direction);	Unsets port mirroring.
int (*hw_if_unset_portmirror) (struct hsl_if *ifp, struct hsl_if *ifp2, enum hal_port_mirror_direction direction);	Sets port selection criteria for aggregator.
int (*hw_if_lacp_psc_set) (struct hsl_if *ifp, int psc);	Adds an aggregator.
int (*hw_if_lacp_agg_add) (struct hsl_if *ifp, int agg_type);	Deletes an aggregator.
int (*hw_if_lacp_agg_del) (struct hsl_if *ifp);	Attaches a port to aggregator.
int (*hw_if_lacp_agg_port_attach) (struct hsl_if *agg_ifp,struct hsl_if *port_ifp);	Detaches a port from aggregator.
int (*hw_if_lacp_agg_port_detach) (struct hsl_if *agg_ifp,struct hsl_if *port_ifp);	Creates an MPLS interfac
void *(*hw_mpls_if_configure) (struct hsl_if *ifp, void *data);	Deletes an MPLS interface

Defintion

```
struct hsl_ifmgr_hw_callbacks
{
  int (*hw_if_init) (void);
  /* Interface manager hardware deinitialization. */
  int (*hw_if_deinit) (void);
  /* Dump. */
  void (*hw_if_dump) (struct hsl_if *ifp);
  /* Unregister L2 port.
     Parameters:
     IN -> ifp - interface pointer
   Returns:
     0 on success
     < 0 on error
  */
  int (*hw_l2_unregister) (struct hsl_if *ifp);
  /* Set L2 port flags.
     Parameters:
     IN -> ifp - interface pointer
     IN -> flags - flags
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_l2_if_flags_set) (struct hsl_if *ifp, unsigned long flags);
  /* Unset L2 port flags.
     Parameters:
     IN -> ifp - interface pointer
     IN -> flags - flags
     Returns:
     0 on success
     < 0 on error
  */
 int (*hw_l2_if_flags_unset) (struct hsl_if *ifp, unsigned long flags);
  /* Set packet types acceptable from this port.
     Parameters:
     IN -> ifp - interface pointer
     IN -> pkt_flags
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_packet_types_set) (struct hsl_if *ifp, unsigned long pkt_flags);
  /* Unset packet types acceptable from this port.
     Parameters:
     IN -> ifp - interface pointer
     IN -> pkt_flags
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_packet_types_unset) (struct hsl_if *ifp, unsigned long pkt_flags);
  /* Set MTU for interface.
 Parameters:
     IN -> ifp - interface pointer
     IN -> mtu - mtu
     Returns:
```

```
    0 on success
    < 0 on error
 */
 int (*hw_if_mtu_set) (struct hsl_if *ifp, int mtu);
 /*  Add/Remove members for Portbased vlan group
     Paramters :
     IN -> ifp - interface pointer
     IN -> pbitmap - Bitmap for ports to be added/removed
     IN -> status -  operation status for add /remove
     Returns:
     0 on success
     < 0 on error
 */
 int (*hw_if_portbased_vlan) (struct hsl_if *ifp, struct hal_port_map pbitmap);
 /*
     Set cpu port default vlan id
     Parameters :
     IN -> vid - vlan id
     Returns:
     0 on success
< 0 on error
 */
 int (*hw_if_cpu_default_vlan) (int vid);
 /*
     Set wayside port default vlan id
     Parameters :
     IN -> vid - vlan id
     Returns:
     0 on success
     < 0 on error
 */
 int (*hw_if_wayside_default_vlan) (int vid);
 /*
    Preserve ce cos
    Parameters :
    IN - ifp - interface pointer
    Returns :
    0 on success
    < 0 on error
 */
 int (*hw_if_preserve_ce_cos)(struct hsl_if *ifp);
/*
     Set port egress mode
     Parameters :
     IN -> ifp    - interface pointer
     IN -> egress - egress mode
     Returns:
     0 on success
     < 0 on error
 */
```

```
  int (*hw_if_port_egress) (struct hsl_if *ifp,
                            int egress);
  /*  Set Force Vlan
      parameters:
      IN -> ifp - interface pointer
      IN -> vid - VLAN id
      Returns:
      0 on success
      < 0 on error
  */
  int (*hw_if_set_force_vlan) (struct hsl_if *ifp, int vid);
 /*  Set Force Vlan
      parameters:
      IN -> ifp - interface pointer
      IN -> etype - ethernet type
      Returns:
      0 on success
      < 0 on error
  */
  int (*hw_if_set_ether_type) (struct hsl_if *ifp, u_int16_t etype);
  /*  Set Force Vlan
     parameters:
      IN -> ifp    - interface pointer
      IN -> enable - learn_disable enable/disable
      IN -> flag   - indicates set/get for  backend function
      Returns:
      0 on success
      < 0 on error
  */
  int (*hw_if_learn_disable) (struct hsl_if *ifp, int *enable,
                              int flag);
  /*  Set Sw Reset
      Returns:
      0 on success
      < 0 on error
  */
  int (*hw_if_set_sw_reset) (void);
  /* Set MTU for L3 interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> mtu - mtu
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_l3_mtu_set) (struct hsl_if *ifp, int mtu);
  /* Set DUPLEX for interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> duplex - duplex
     Returns:
     0 on success
     < 0 on error
  */
 int (*hw_if_duplex_set) (struct hsl_if *ifp, int duplex);
  /* Set AUTONEGO for interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> autonego - autonego
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_autonego_set) (struct hsl_if *ifp, int autonego);
  /* Set BANDWIDTH for interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> bandwidth - bandwidth
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_bandwidth_set) (struct hsl_if *ifp, u_int32_t bandwidth);
  /* Set HW address for a interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> hwadderlen - address length
     IN -> hwaddr - address
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_hwaddr_set) (struct hsl_if *ifp, int hwaddrlen, u_char *hwaddr);
  /* Set secondary HW addresses for a interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> hwaddrlen - address length
     IN -> num - number of secondary addresses
     IN -> addresses - array of secondary addresses
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_secondary_hwaddrs_set) (struct hsl_if *ifp, int hwaddrlen, int num, u_char
**addresses);
  /* Add secondary HW addresses for a interface.
  Parameters:
     IN -> ifp - interface pointer
     IN -> hwaddrlen - address length
     IN -> num - number of secondary addresses
     IN -> addresses - array of secondary addresses
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_secondary_hwaddrs_add) (struct hsl_if *ifp, int hwaddrlen, int num, u_char
**addresses);
  /* Delete secondary HW addresses for a interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> hwaddrlen - address length
     IN -> num - number of secondary addresses
     IN -> addresses - array of secondary addresses
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_secondary_hwaddrs_delete) (struct hsl_if *ifp, int hwaddrlen, int num,
u_char **addresses);
 /* Create L3 interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> data - system specific data
     Returns:
     HW L3 interface pointer as void *
     NULL on error
  */
  void *(*hw_l3_if_configure) (struct hsl_if *ifp, void *data);
  /* Perform any post configuration. This can typically be done
     after some interface binding is performed.
     Parameters:
     IN -> ifp - interface pointer
     IN -> ifp - interface pointer
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_post_configure) (struct hsl_if *ifpp, struct hsl_if *ifpc);
  /* Perform any pre unconfiguration. This can typically be done
     before some interface unbinding is performed.
 Parameters:
     IN -> ifp - interface pointer
     IN -> ifp - interface pointer
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_pre_unconfigure) (struct hsl_if *ifpp, struct hsl_if *ifpc);
  /* Delete L3 interface.
     Parameters:
     IN -> ifp - interface pointer
     Returns:
```

```
    0 on success
    < 0 on error
 */
 int (*hw_l3_if_unconfigure) (struct hsl_if *ifp);
 /* Set switching type for a port.
    Parameters:
    IN -> ifp
Returns:
    0 on success
    < 0 on error
 */
 int (*hw_set_switching_type) (struct hsl_if *ifp, hsl_IfSwitchType_t type);
 /* Set L3 port flags.
    Parameters:
    IN -> ifp - interface pointer
    IN -> flags - flags
    Returns:
    0 on success
    < 0 on error
 */
 int (*hw_l3_if_flags_set) (struct hsl_if *ifp, unsigned long flags);
 /* Unset L3 port flags.
    Parameters:
    IN -> ifp - interface pointer
    IN -> flags - flags
    Returns:
    0 on success
    < 0 on error
 */
 int (*hw_l3_if_flags_unset) (struct hsl_if *ifp, unsigned long flags);
/* Add a IP address to the interface.
   Parameters:
    IN -> ifp - interface pointer
    IN -> prefix - interface address and prefix
    IN -> flags - flags
    Returns:
    0 on success
    < 0 on error
 */
 int (*hw_l3_if_address_add) (struct hsl_if *ifp,
                              hsl_prefix_t *prefix, u_char flags);
 /* Delete a IP address from the interface.
    Parameters:
    IN -> ifp - interface pointer
    IN -> prefix - interface address and prefix
    Returns:
    0 on success
    < 0 on error
 */
int (*hw_l3_if_address_delete) (struct hsl_if *ifp,
                                  hsl_prefix_t *prefix);
  /* Get interface MAC counters.
     Parameters:
     INOUT -> ifp - interface pointer
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_get_counters) (struct hsl_if *ifp);
/* Clear the Interface Counters.
   Parameter
   INOUT -> ifp - interface pointer
   Returns:
   0 in success
   < 0 in error
*/
  int (*hw_if_clear_counters) (struct hsl_if *ifp);
/* MDIX crossover.
   Parameter
 IN -> ifp - interface pointer
   IN -> mdix - MDIX crossover value
   Returns:
   0 in success
   < 0 in error
*/
  int (*hw_if_mdix_set) (struct hsl_if *ifp, int mdix);
#ifdef HAVE_L3
  /* Bind a interface to a FIB
     Parameters:
     IN -> ifp - interface pointer
     IN -> fib_id - FIB id
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_l3_if_bind_fib) (struct hsl_if *ifp,
                                  hsl_fib_id_t fib_id);
  /* Unbind a interface from a FIB
     Parameters:
     IN -> ifp - interface pointer
     IN -> fib_id - FIB id
 Returns:
     0 on success
     < 0 on error
  */
  int (*hw_l3_if_unbind_fib) (struct hsl_if *ifp,
                                  hsl_fib_id_t fib_id);
  /* Init port mirroring.
     Parameters:
     void
     Returns:
     0 on success
     < 0 on error
  */
#endif /* HAVE_L3 */
  int (*hw_if_init_portmirror) (void);
  /* Deinit port mirroring.
     Parameters:
     void
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_deinit_portmirror) (void)
  /* Set port mirroring.
     Parameters:
     IN -> ifp - mirroring interface
     IN -> ifp - mirrored  interface
     IN -> direction - mirrored traffic direction
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_set_portmirror) (struct hsl_if *ifp, struct hsl_if *ifp2, enum 
hal_port_mirror_direction direction);
  /* Unset port mirroring.
     Parameters:
     IN -> ifp - mirroring interface
     IN -> ifp - mirrored  interface
     IN -> direction - mirrored traffic direction
 Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_unset_portmirror) (struct hsl_if *ifp, struct hsl_if *ifp2, enum 
hal_port_mirror_direction direction);
#ifdef HAVE_LACPD
  /* Set port selection criteria for aggregator.
     Parameters:
     IN -> ifp - aggregator interface
     IN -> psc - port selection criteria.
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_lacp_psc_set) (struct hsl_if *ifp, int psc);
  /* Add aggregator.
     Parameters:
     IN -> agg_name - aggregator name.
     IN -> agg_mac  - aggregator hw address.
     IN -> agg_type - aggregator type.
     Returns:
     0 on success
     < 0 on error
 */
  int (*hw_if_lacp_agg_add) (struct hsl_if *ifp, int agg_type);
  /* Delete aggregator.
     Parameters:
     IN -> ifp - aggregator interface.
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_lacp_agg_del) (struct hsl_if *ifp);
  /* Attach port to aggregator.
     Parameters:
     IN -> agg_ifp - aggregator interface.
     IN -> port_ifp - port interface.
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_lacp_agg_port_attach) (struct hsl_if *agg_ifp,struct hsl_if *port_ifp);
  /* Detach port from aggregator.
 Parameters:
     IN -> agg_ifp - aggregator interface.
     IN -> port_ifp - port interface.
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_if_lacp_agg_port_detach) (struct hsl_if *agg_ifp,struct hsl_if *port_ifp);
#endif /* HAVE_LACPD */
#ifdef HAVE_MPLS
  /* Create MPLS interface.
     Parameters:
     IN -> ifp - interface pointer
     IN -> data - system specific data
     Returns:
     HW MPLS L3 interface pointer as void *
     NULL on error
  */
  void *(*hw_mpls_if_configure) (struct hsl_if *ifp, void *data);
  /* Delete MPLS interface.
  Parameters:
     IN -> ifp - interface pointer
     Returns:
     0 on success
     < 0 on error
  */
  int (*hw_mpls_if_unconfigure) (struct hsl_if *ifp);
```

```
#endif /* HAVE_MPLS */
};
```

hsl_if_notifier_events

This enum is defined in libhsl/common/hsl_ifmgr.h:

```
enum hsl_if_notifier_events
  {
    HSL_IF_EVENT_IFNEW          = 100,
    HSL_IF_EVENT_IFDELETE       = 101,
    HSL_IF_EVENT_IFFLAGS        = 102,
    HSL_IF_EVENT_IFNEWADDR      = 103,
    HSL_IF_EVENT_IFDELADDR      = 104,
    HSL_IF_EVENT_IFMTU          = 105,
    HSL_IF_EVENT_IFHWADDR       = 106,
    HSL_IF_EVENT_IFDUPLEX       = 107,
    HSL_IF_EVENT_IFAUTONEGO     = 108,
    HSL_IF_EVENT_IFBANDWIDTH    = 109,
    HSL_IF_EVENT_IFARPAGEINGTIMEOUT = 110,
    HSL_IF_EVENT_IF_UPDADDR     = 111,
    HSL_IF_EVENT_STP_REFRESH    = 112,
    HSL_IF_EVENT_MDIX           = 113,
    HSL_IF_EVENT_PORTBASED_VLAN = 114,
    HSL_IF_EVENT_PORT_EGRESS    = 115,
    HSL_IF_EVENT_CPU_DEFAULT_VLAN = 116,
    HSL_IF_EVENT_FORCE_VLAN     = 117,
    HSL_IF_EVENT_ETHERTYPE      = 118,
    HSL_IF_EVENT_LEARN_DISABLE  = 119,
    HSL_IF_EVENT_SW_RESET       = 120,
    HSL_IF_EVENT_WAYSIDE_DEFAULT_VLAN = 121,
    HSL_IF_EVENT_PRESERVE_CE_COS = 122,
    HSL_IF_EVENT_FPWINDOW_EXPIRY = 123,
};
```

hsl_nh_entry

This data structure helps manage nexthop entries. It is defined in libhsl/L3/hsl_fib.h:

Type	Definition
ifp	Interface
l2_ifp	Layer2 interface
HSL_ETHER_ALEN	Ethernet address
flags	Flags, including: HSL_NH_ENTRY_VALID (1 << 0) HSL_NH_ENTRY_STATIC (1 << 1) HSL_NH_ENTRY_DEL_IN_PROGRESS (1 << 2) HSL_NH_ENTRY_DEPENDENT (1 << 3) HSL_NH_ENTRY_PROXY (1 << 4) HSL_NH_ENTRY_BLACKHOLE (1 << 5)

Type	Definition
ext_flags	Flags, including: HSL_NH_ENTRY_EFLAG_IN_HW (1 << 0) HSL_NH_TYPE_IP 0 HSL_NH_TYPE_IPV6 1 HSL_NH_TYPE_MPLS 2
system_info	Hardware specific information for a nexthop.
refcnt	Number of routes, pointing to this nexthop.
prefix_tree	Tree of prefix pointers dependent on a nexthop.
ilm_list	List of ILM (incoming label map) entries dependent on a nexthop.
aliveCounter	Liveliness counter
rn	Pointer to parent tree node

Definition

```
struct hsl_nh_entry
{
  struct hsl_if *ifp;                 /* Interface. */
  struct hsl_if *l2_ifp;              /* Layer2 interface */
  u_char mac[HSL_ETHER_ALEN];         /* Ethernet address. */
  u_char flags;                       /* Flags. */
#define HSL_NH_ENTRY_VALID               (1 << 0)
#define HSL_NH_ENTRY_STATIC              (1 << 1)
#define HSL_NH_ENTRY_DEL_IN_PROGRESS     (1 << 2)
#define HSL_NH_ENTRY_DEPENDENT           (1 << 3)
#define HSL_NH_ENTRY_PROXY               (1 << 4)
#define HSL_NH_ENTRY_BLACKHOLE           (1 << 5)
  u_char ext_flags;                       /* Flags. */
#define HSL_NH_ENTRY_EFLAG_IN_HW         (1 << 0)
#define HSL_NH_TYPE_IP                0
#define HSL_NH_TYPE_IPV6              1
#define HSL_NH_TYPE_MPLS              2
  u_char nh_type;
  void *system_info;                  /* Hardware specific info for this nexthop. */
  u_int32_t refcnt;                   /* Number of routes, pointing to this nexthop */
  struct hsl_avl_tree *prefix_tree;  /* Tree of prefix pointers dependent on this NH. */
#ifdef HAVE_MPLS
  struct hsl_mpls_ilm_entry *ilm_list; /* List of ILM entries dependent on this NH */
#ifdef HAVE_MPLS_VC
  struct hsl_mpls_vpn_vc *vpn_vc_list;
#endif /* HAVE_MPLS_VC */
#endif /* HAVE_MPLS */
  u_int32_t aliveCounter;             /* Liveliness counter. */
 struct hsl_route_node *rn;        /* Pointer to parent tree node */
  struct hsl_nh_entry *next;
};
```

hsl_nh_entry_list_node

This data structure helps manage nexthop entry list nodes. It is defined in libhsl/L3/hsl_fib.h:

Definition

```
struct hsl_nh_entry_list_node
{
  struct hsl_nh_entry *entry;
  struct hsl_nh_entry_list_node *next;
};
```

hsl_nh_if_list_node

This data structure helps manage nexthop interface list nodes. It is defined in libhsl/L3/hsl_fib.h:

Type	Definition
ifp	Interface information
*next	Next node in linked list

Definition

```
struct hsl_nh_if_list_node
{
  struct hsl_if *ifp;               - Interface information (See Interface manager).
  struct hsl_nh_if_list_node *next; - Next node in linked list.
};
```

hsl_prefix_entry

This data structure helps manage prefix entries. It is defined in libhsl/L3/hsl_fib.h:

Type	Definition
flags	Flags, including: HSL_PREFIX_ENTRY_IN_HW (1 << 0) HSL_PREFIX_ENTRY_EXCEPTION (1 << 1)
system_info	System specific info for this prefix
nhcount	Reference count (that is, total nexthops)
nhlist	List of nexthops
ifcount	Reference count for interface routes
iflist	List of ifps as nexthops

Definition

```
struct hsl_prefix_entry
{
  u_char flags;                      /* Flags. */
#define HSL_PREFIX_ENTRY_IN_HW          (1 << 0)
#define HSL_PREFIX_ENTRY_EXCEPTION   (1 << 1)
  void *system_info;                 /* System specific info for this prefix. */
  u_char nhcount;                    /* Ref count, really is the total nexthops. */
  struct hsl_nh_entry_list_node *nhlist;    /* List of NHs. */
  u_char ifcount;                    /* Ref count, for interface routes. */
  struct hsl_nh_if_list_node *iflist;       /* List of ifps as NH. */
};
```

hsl_route_table

This data structure helps manage routing table functions. It is defined in libhsl/common/hsl_table.h:

Definition

```
struct hsl_route_table
{
  struct hsl_route_node *top;
  /* Table identifier. */
  u_int32_t id;
};
```

hsl_route_node

This data structure helps manage router node functions. It is defined in libhsl/common/hsl_table.h:

Type	Definition
hsl_route_node	Link, including: l_left link[0] l_right link[1]
p	Actual prefix of this radix
is_ecmp	Flag for ECMP
table	Tree link
*paren	Tree link
lock	Lock of this radix
info	Each node of route

Definition

```
struct hsl_route_node
{
  /* DO NOT MOVE the first 2 pointers. They are used for memory
     manager as well */
  struct hsl_route_node *link[2];
#define l_left   link[0]
#define l_right  link[1]
  /* Actual prefix of this radix. */
  hsl_prefix_t p;
  /* Flag for ECMP */
  HSL_BOOL is_ecmp;
  /* Tree link. */
  struct hsl_route_table *table;
  struct hsl_route_node *parent;
  /* Lock of this radix */
  u_int32_t lock;
  /* Each node of route. */
  void *info;
};
```

hsl_bcm_rx_queue

This data structure helps manage BCM received queue functions. It is defined in hsl/broadcom/hsl_bcm_pkt.h:

Type	Definition
pkt_queue	BCM packet queue of aligned bcm_pkt_t.
total	Total queue size.
head	Head of queue.
tail	Tail of queue.
count	Number of packets in queue.
drop	Number of dropped packets.
pkt_thread	Packet execution thread.
pkt_sem	Packet semaphore.
thread_exit	If one, exits packet processing.

Definition

```
struct hsl_bcm_rx_queue
{
  u_char *pkt_queue;                 /* BCM Packet queue of aligned bcm_pkt_t. */
  int total;                         /* Total queue size. */
  int head;                          /* Head of queue. */
  int tail;                          /* Tail of queue. */
  int count;                         /* Number of packets in queue. */
  int drop;                          /* Number of dropped packets. */
  struct sal_thread_s *pkt_thread;   /* Packet execution thread. */
  ipi_sem_id pkt_sem;                /* Packet semaphore. */
  int thread_exit;                   /* If 1, exit packet processing. */
};
```

hsl_bcm_tx_queue

This data structure helps manage BCM transmitted queue functions. It is defined in hsl/broadcom/hsl_bcm_pkt.h:

Type	Definition
*pkt_list	BCM packet list of aligned bcm_pkt_t.
*free_pkt_list	Free list of aligned bcm_pkt_t.
total	Total list size.
count	Current count.
pkt_sem	Semaphore to protect this list.

Definition

```
struct hsl_bcm_tx_queue
{
  u_char *pkt_list;                   /* BCM packet list of aligned bcm_pkt_t. */
  u_char *free_pkt_list;              /* Free list of aligned bcm_pkt_t. */
  int total;                          /* Total list size. */
  int count;                          /* Current count. */
  ipi_sem_id pkt_sem;                 /* Semaphore to protect this list. */
};
```

hsl_eth_tx_drv_netpool

This data structure helps manage Ethernet transmit netpool functions. It is defined in hsl/broadcom/vxworks/L3/hsl_eth_drv.h:

Definition

```
struct hsl_eth_tx_drv_netpool
{
  int initialized;                /* Initialized. */
  M_CL_CONFIG blks;               /* Num
  CL_DESC cl;
  CL_POOL_ID clpool_id;
  NET_POOL_ID netpool_id;
};
```

HSL Callbacks

HSL callbacks form a separation among the hardware, operating system and the control plane software. Callbacks have corresponding API functions in the HAL. These function trigger corresponding callbacks. All HSL callbacks are described in this subsection:

Configuration Callbacks

The following table displays the configuration callbacks.

Table 1:

Callbacks	Description
`bridge_init`	Create bridge
`bridge_deinit`	Delete bridge
`set_age_timer`	Set bridge MAC ageing time
`set_learning`	Enable/disable learning on bridge.
`add_port_to_bridge`	Add port to bridge
`delete_port_from_bridge`	Delete port from bridge
`set_proto_dest_mac`	Add user-defined MAC for Protocols

FIB Hardware Multicast Callbacks

The following table displays the FIB (forwarding information base) multicast callbacks for the system hardware.

Table 2:

Callbacks	Description
`hw_ipv4_mc_init`	Initialize multicast routing in the hardware
`hw_ipv4_mc_deinit`	Deinitialize multicast routing in the hardware
`hw_ipv4_mc_route_add`	Add a multicast route to the hardware
`hw_ipv4_mc_route_del`	Delete a multicast route from the hardware
`hw_ipv4_mc_sg_stat`	Get multicast route usage statistics
`hw_ipv4_mc_vif_add`	Add a multicast interface to the hardware
`hw_ipv4_mc_vif_del`	Delete a multicast interface from the hardware

FIB OS Multicast Callbacks

The following table displays the FIB multicast callbacks for operating systems.

Table 3:

Callbacks	Description
`os_ipv4_mc_pim_init`	Initialize PIM in the OS
`os_ipv4_mc_pim_deinit`	Deinitialize PIM in the OS
`os_ipv4_mc_route_add`	Add a multicast route in the OS
`os_ipv4_mc_route_del`	Delete a multicast route from the OS
`os_ipv4_mc_vif_add`	Add a multicast interface to the OS
`os_ipv4_mc_vif_del`	Delete a multicast interface from the OS

Flow Control Callbacks

The following table displays the flow control callbacks.

Table 4:

Callbacks	Description
`set_flowcontrol`	Enable or disable flow control messages on port
`get_flowcontrol_statistics`	Get flow control messages statistics

Forwarding Database Callbacks

The following table displays the forwarding database callbacks.

Table 5:

Callbacks	Description
`add_fdb`	Add a MAC entry to forwarding database (FDB)
`delete_fdb`	Delete a MAC entry from FDB
`get_uni_fdb`	Get dynamic unicast MAC entries from FDB
`flush_port_fdb`	Flush FDB for specific port
`flush_fdb_by_mac`	Delete specific MAC from FDB

IGMP Snooping Callbacks

The following table displays the IGMP (Internet Group Management Protocol) callbacks.

Table 6:

Callbacks	Description
`enable_igmp_snooping`	Enable IGMP snooping
`disable_igmp_snooping`	Disable IGMP snooping
enable_igmp_snooping_port	Enable IGMP snooping on port
disable_igmp_snooping_port	Disable IGMP snooping on port

Hardware Callbacks

The following table displays the hardware callbacks.

Table 7:

Callback	Description
hw_fib_init	Initialize the hardware FIB.
hw_fib_deinit	De-initialize the hardware FIB.
hw_fib_dump	Show the hardware FIB entries.
hw_prefix_add	Add a prefix to the hardware.
hw_prefix_add_exception	Add an exception prefix to the hardware. All packets matching this prefix will be trapped to CPU.
hw_prefix_delete	Delete a prefix from the hardware.
hw_nh_add	Add a next hop to the hardware.
hw_nh_delete	Delete a next hop from the hardware.
hw_nh_hit	Check next-hop usage in the hardware.
hw_add_connected_route	Add an IPv4 connected route to the hardware.
hw_delete_connected_route	Delete an IPv4 connected from the hardware.
hw_get_max_multipath	Get maximum number of multipaths from hardware.

MLD Snooping Callbacks

The following table displays the MLD (Multicast Listener Discovery) snooping callbacks.

Table 8:

Callbacks	Description
enable_mld_snooping	Enable MLD snooping
disable_mld_snooping	Disable MLD snooping

OS Callbacks

The following table displays the operating system callbacks.

Table 9:

Callback	Description
os_fib_init	Initialize the OS FIB.
os_fib_deinit	De-initialize the OS FIB.

Table 9:

Callback	Description
`os_fib_dump`	Show the OS FIB entries.
`os_prefix_add`	Add a prefix to the OS.
`os_prefix_add_if`	Add a local address to the OS.
`os_prefix_delete`	Delete a prefix from the OS.
`os_prefix_delete_if`	Delete a local address from the OS.
`os_nh_add`	Add a next-hop entry.
`os_nh_delete`	Delete a next-hop entry.

Rate Limiting Callbacks

The following table displays the rate limiting callbacks.

Table 10:

Callbacks	Description
`ratelimit_bcast`	Set broadcast rate limiting
`ratelimit_mcast`	Set multicast rate limiting
`ratelimit_dlf_bcast`	Set unknown destination rate limiting
`ratelimit_bcast_discards_get`	Get number of dropped (rate-limited) broadcasts
`ratelimit_mcast_discards_get`	Get number of dropped (rate-limited) multicast
`ratelimit_dlf_bcast_discards_get`	Get number of dropped (rate-limited) unknown destination packets

VLAN Callbacks

The following table displays the VLAN callbacks.

Table 11:

Callbacks	Description
`add_vlan`	Add VLAN to bridge
`delete_vlan`	Delete VLAN from bridge
`set_vlan_port_type`	Set VLAN port type (Access/Trunk/Hybrid)
`set_default_pvid`	Set default VLAN ID on port

Table 11:

Callbacks	Description
`add_vlan_to_port`	Add VLAN to port
`delete_vlan_from_port`	Remove VLAN from port
set_mac_prio_over	Set MAC priority override
set_dot1q_state	Disable DOT1Q
set_default_pvid	Set default PVID
`vlan_mac_classifier_add`	Add a MAC-based VLAN classifier
`vlan_ipv4_classifier_add`	Add a subnet-based VLAN classifier
`vlan_proto_classifier_add`	Add a protocol-based VLAN classifier
`vlan_mac_classifier_delete`	Delete a MAC-based VLAN classifier
`vlan_ipv4_classifier_delete`	Delete a subnet-based VLAN classifier
`vlan_proto_classifier_delete`	Delete a protocol-based VLAN classifier

xSTP Callbacks

The following table displays the XSTP callbacks.

Table 12:

Callbacks	Description
`set_stp_port_state`	Set port STP state: blocked, listening, learning, forwarding
`add_instance`	Add a bridge instance (MSTP)
`delete_instance`	Remove a bridge instance.
`add_vlan_to_instance`	Add a VLAN to an instance
`delete_vlan_from_instance`	Delete a VLAN from an instance

General API Functions

The following table and subsection list the general API functions for HSL.

Table 13: General API functions

Functions	Description
hsl_ifmgr_dump	Dumps an HSL interface manager.
hsl_ifmgr_init	Initializes the HSL interface manager.
hsl_ifmgr_deinit	De-initializes the HSL interface manager.
hsl_ifmgr_notify_chain_register	Notifies the interface manager of any new chain registration.
hsl_ifmgr_notify_chain_unregister	Notifies the interface manager of any new chain unregistrations.
hsl_ifmgr_lock_children	Locks all children.
hsl_ifmgr_unlock_children	Unlocks all children.
hsl_ifmgr_lock_parents	Locks all parents.
hsl_ifmgr_unlock_parents	Unlocks all parents.
hsl_ifmgr_set_os_callbacks	Registers an OS-specific callback.
hsl_ifmgr_unset_os_callbacks	Unregisters an OS-specific callback.
hsl_ifmgr_set_hw_callbacks	Registers an hardware-specific callback.
hsl_ifmgr_unset_hw_callbacks	Unregister an hardware-specific callback.
hsl_sock_nh_event	
hsl_msg_nh_resolve	

hsl_ifmgr_dump

This function “dumps” an HSL interface manager.

Syntax

```
void
hsl_ifmgr_dump (void)
```

Input Parameters

None

Output Parameters

None

Return Values

None

hsl_ifmgr_init

This function initializes the HSL interface manager.

Syntax

```
int
hsl_ifmgr_init (void);
```

Input Parameters

None

Output Parameters

None

Return Values

None

hsl_ifmgr_deinit

This function deinitializes the HSL interface manager.

Syntax

```
int
hsl_ifmgr_deinit (void)
```

Input Parameters

None

Output Parameters

None

Return Values

None

hsl_ifmgr_notify_chain_register

This function notifies the interface manager of any new chain registration.

Syntax

```
int
hsl_ifmgr_notify_chain_register (struct hsl_if_notifier_chain *new)
```

Input Parameters

`new`	A new chain registration.

Output Parameters

None

Return Values

hsl_ifmgr_notify_chain_unregister

This function notifies the interface manager of any new chain unregistrations.

Syntax

```
int
hsl_ifmgr_notify_chain_unregister (struct hsl_if_notifier_chain *old)
```

Input Parameters

`old`	An old chain registration.

Output Parameters

None

Return Values

hsl_ifmgr_lock_children

This function locks all children.

Syntax

```
void
hsl_ifmgr_lock_children (struct hsl_if *ifp)
```

Input Parameters

`ifp`	Interface port.

Output Parameters

None

Return Values

None

hsl_ifmgr_unlock_children

This function unlocks all children.

Syntax

```
void
hsl_ifmgr_unlock_children (struct hsl_if *ifp)
```

Input Parameters

`ifp`	Interface port.

Output Parameters

None

Return Values

None

hsl_ifmgr_lock_parents

This function locks all parent.

Syntax

```
void
hsl_ifmgr_lock_parents (struct hsl_if *ifp)
```

Input Parameters

ifp	Interface port.

Output Parameters

None

Return Values

None

hsl_ifmgr_unlock_parents

This function unlocks all children.

Syntax

```
void
hsl_ifmgr_unlock_parents (struct hsl_if *ifp)
```

Input Parameters

ifp	Interface port.

Output Parameters

None

Return Values

None

hsl_ifmgr_set_os_callbacks

This function registers an OS-specific callback.

Syntax

```
int
hsl_ifmgr_set_os_callbacks (struct hsl_ifmgr_os_callbacks *cb)
```

Input Parameters

None

Output Parameters

`cb`	Specific callback

Return Values

hsl_ifmgr_unset_os_callbacks

This function unregisters an OS-specific callback.

Syntax

```
int
hsl_ifmgr_unset_os_callbacks (void)
```

Input Parameters

None

Output Parameters

None

Return Values

hsl_ifmgr_set_hw_callbacks

This function registers an hardware-specific callback.

Syntax

```
int
hsl_ifmgr_set_hw_callbacks (struct hsl_ifmgr_hw_callbacks *cb)
```

Input Parameters

None

Output Parameters

`cb`	Specific callback

Return Values

hsl_ifmgr_unset_hw_callbacks

This function unregisters an hardware-specific callback.

Syntax

```
int
hsl_ifmgr_unset_hw_callbacks (void)
```

Input Parameters

None

Output Parameters

None

Return Values

hsl_sock_nh_event

This function is used to process the NH events.

Syntax

```
    int
    hsl_sock_nh_event (int cmd, void *param1, void *param2);
```

Input Parameters

cmd	Command descriptor.
param1	Parameter one.
param2	Parameter two.

Output Parameters

None

Returns

HSL_SUCCESS on success.

-1

hsl_msg_nh_resolve

This function decodes IPv4 NH resolve messages.

Syntax

```
    int
    hsl_msg_nh_resolve (struct socket *sock, void *param1, void *param2)
```

Input Parameters

sock	Socket descriptor.
param1	Parameter one.
param2	Parameter two.

Output Parameters

None

Returns

0

-1

Interface API Functions

The following table and subsection includes the interface API functions for HSL.

Table 14: Interface API functions

Functions	Description
hsl_ifmgr_lookup_by_index	Finds an interface by index.
hsl_ifmgr_lookup_by_name	Finds an interface by name.
hsl_ifmgr_set_acceptable_packet_types	Sets the packet types that can be accepted for an interface.
hsl_ifmgr_unset_acceptable_packet_types	Unsets the packet types that were accepted for an interface.
hsl_ifmgr_isbound	Indicates whether an interface is bound to another.
hsl_ifmgr_bind	Binds an interface with an interface pointer.
hsl_ifmgr_bind2	Binds an interface with an interface pointer.
hsl_ifmgr_unbind	Unbinds an interface from an interface pointer.
hsl_ifmgr_unbind2	Unbinds an interface from an interface pointer.
hsl_ifmgr_bindings_add	Binds an interface to set of child ports.
hsl_ifmgr_bindings_remove_all	Remove all bindings.
hsl_ifmgr_set_flags2	Sets flags for an interface that is given an interface pointer.
hsl_ifmgr_set_flags	Sets flags for an interface.
hsl_ifmgr_unset_flags2	Unsets flags for an interface that is given an interface pointer.
hsl_ifmgr_unset_flags	Unsets flags for an interface.
hsl_ifmgr_create_interface	Registers an interface with an interface manager.
hsl_ifmgr_delete_interface	Deletes an interface from an interface manager.
hsl_ifmgr_delete_interface_api	Removes an interface from the interface manager.
hsl_ifmgr_set_mtu	Sets MTU for an interface.
hsl_ifmgr_set_duplex	Sets duplex for an interface.
hsl_ifmgr_set_autonego	Sets auto-negotiations for an interface.
hsl_ifmgr_set_bandwidth	Sets bandwidth for an interface.
hsl_ifmgr_set_hwaddr	Sets the hardware address for an interface.
hsl_ifmgr_set_arp_ageing_timeout	Sets the ARP ageing time out value for an interface.

Table 14: Interface API functions (Continued)

Functions	Description
hsl_ifmgr_get_if_counters	Sets a port to a router port
hsl_ifmgr_collect_if_stat	Processes interface statistics collection.

hsl_ifmgr_lookup_by_index

This function finds an interface by index.

Note: The interface reference count must be decremented after calling this function.

Syntax

```
struct hsl_if *
hsl_ifmgr_lookup_by_index (hsl_ifIndex_t ifindex)
```

Input Parameters

`ifindex`	Interface index.

Output Parameters

None

Return Values

hsl_ifmgr_lookup_by_name

This function finds an interface by name.

Note: The interface reference count must be decremented after calling this function.

Syntax

struct hsl_if *

hsl_ifmgr_lookup_by_name (char *name)

Input Parameters

`name`	Interface name

Output Parameters

None

Return Values

hsl_ifmgr_set_acceptable_packet_types

This function sets the packet types that can be accepted for an interface.

Syntax

```
void
```

```
hsl_ifmgr_set_acceptable_packet_types (struct hsl_if *ifp, u_int32_t pkt_flags)
```

Input Parameters

`ifp`	Interface port.
`pkt_flags`	Packet flags.

Output Parameters

None

Return Values

None

hsl_ifmgr_unset_acceptable_packet_types

This function unsets the packet types that were accepted for an interface.

Syntax

```
void
hsl_ifmgr_unset_acceptable_packet_types (struct hsl_if *ifp, u_int32_t pkt_flags)
```

Input Parameters

`ifp`	Interface port.
`pkt_flags`	Packet flags.

Output Parameters

None

Return Values

None

hsl_ifmgr_isbound

This function indicates whether an interface is bound to another.

Syntax

```
HSL_BOOL
hsl_ifmgr_isbound (struct hsl_if *ifpp, struct hsl_if *ifpc)
```

Input Parameters

`ifpp`	Interface pointer
`ifpc`	

Output Parameters

None

Return Values

hsl_ifmgr_bind

This function binds an interface with an interface index.

Syntax

```
int
hsl_ifmgr_bind (hsl_ifIndex_t parentIfindex, hsl_ifIndex_t childIfindex)
```

Input Parameters

`parentIfindex`	Parent interface index.
`childIfindex`	Child interface index.

Output Parameters

None

Return Values

hsl_ifmgr_bind2

This function binds an interface with an interface pointer.

Syntax

```
int
hsl_ifmgr_bind2 (struct hsl_if *ifpp, struct hsl_if *ifpc)
```

Input Parameters

`ifpp`	Interface pointer
`ifpc`	

Output Parameters

None

Return Values

hsl_ifmgr_unbind

This function unbinds an interface from an interface pointer.

Syntax

```
int
hsl_ifmgr_unbind2 (struct hsl_if *ifpp, struct hsl_if *ifpc)
```

Input Parameters

`ifpp`	Interface pointer
`ifpc`	

Output Parameters

None

Return Values

hsl_ifmgr_unbind2

This function unbinds an interface using the interface pointer.

Syntax

```
int
hsl_ifmgr_unbind2 (struct hsl_if *ifpp, struct hsl_if *ifpc)
```

Input Parameters

`ifpp`	Interface pointer
`ifpc`	

Output Parameters

None

Return Values

hsl_ifmgr_bindings_add

This function binds an interface to set of child ports.

Syntax

```
int
hsl_ifmgr_bindings_add (struct hsl_if *ifpp, int num,
                        hsl_ifIndex_t *ifindexes)
```

Input Parameters

`ifpp`	Interface port pointer.
`num`	

Output Parameters

`ifindexes`	Array of ports to add.

Return Values

hsl_ifmgr_bindings_remove_all

This function remove all bindings.

Syntax

```
int
hsl_ifmgr_bindings_remove_all (struct hsl_if *ifpp)
```

Input Parameters

`ifpp`	Interface port pointer.

Output Parameters

None

Return Values

hsl_ifmgr_set_flags2

This function sets flags for an interface that is given an interface pointer.

Syntax

```
int
hsl_ifmgr_set_flags2 (struct hsl_if *ifp, u_int32_t flags)
```

Input Parameters

`ifp`	Interface pointer.
`flags`	

Output Parameters

None

Return Values

hsl_ifmgr_set_flags

This function sets flags for an interface.

Syntax

```
int
hsl_ifmgr_set_flags (char *name, hsl_ifIndex_t ifindex, u_int32_t flags)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.
`flags`	

Output Parameters

None

Return Values

hsl_ifmgr_unset_flags2

This function unsets flags for an interface that is given an interface pointer.

Syntax

```
int
hsl_ifmgr_unset_flags2 (struct hsl_if *ifp, u_int32_t flags)
```

Input Parameters

`ifp`	Interface pointer.
`flags`	

Output Parameters

None

Return Values

hsl_ifmgr_unset_flags

This function unsets flags for an interface.

Syntax

```
int
hsl_ifmgr_unset_flags (char *name, hsl_ifIndex_t ifindex, u_int32_t flags)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.
`flags`	

Output Parameters

None

Return Values

hsl_ifmgr_create_interface

This function registers an interface with an interface manager.

Syntax

```
int
_hsl_ifmgr_create_interface(struct hsl_if *ifp_params,
                            struct hsl_if **new_ifp,
                            HSL_BOOL send_notification,
                            HSL_BOOL allocated_params,
                            HSL_BOOL create_proc_entry)
```

Input Parameters

`ifp_params` Interface parameters.

`send_notification`

Indicates whether or not to send a notification to protocol modules.

`allocated_params`

`create_proc_entry`

Output Parameters

`new_ifp`	Mirrored from the port.

Return Values

hsl_ifmgr_delete_interface

This function removes an interface from the interface manager.

Syntax

```
int
hsl_ifmgr_delete_interface(struct hsl_if *ifp,
                           HSL_BOOL send_notification)
```

Input Parameters

`ifp`	Interface pointer.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_delete_interface_api

This function removes an interface from the interface manager.

Syntax

```
int
hsl_ifmgr_delete_interface_api(hsl_ifIndex_t ifindex)
```

Input Parameters

`ifindex`	Interface index.

Output Parameters

None

Return Values

hsl_ifmgr_set_mtu

This function sets MTU for an interface.

Syntax

```
int
hsl_ifmgr_set_mtu (hsl_ifIndex_t ifindex, int mtu, HSL_BOOL send_notification)
```

Input Parameters

`ifindex`	Interface index.
`mtu`	Maximum transmission unit.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_set_duplex

This function sets the duplex mode for an interface.

Syntax

```
int
hsl_ifmgr_set_duplex (hsl_ifIndex_t ifindex, int duplex, HSL_BOOL send_notification)
```

Input Parameters

`ifindex`	Interface index.
`duplex`	Duplex mode.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_set_autonego

This function sets auto-negotiations for an interface.

Syntax

```
int
```

```
hsl_ifmgr_set_autonego (hsl_ifIndex_t ifindex, int autonego, HSL_BOOL
send_notification)
```

Input Parameters

`ifindex`	Interface index.
`autonego`	Enable or disable auto-negotiation.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_set_bandwidth

This function sets bandwidth for an interface.

Syntax

```
int
hsl_ifmgr_set_bandwidth (hsl_ifIndex_t ifindex, u_int32_t bandwidth, HSL_BOOL
send_notification)
```

Input Parameters

`ifindex`	Interface index.
`bandwidth`	Bandwidth.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_set_hwaddr

This function sets the hardware address for an interface.

Syntax

```
int
hsl_ifmgr_set_hwaddr (hsl_ifIndex_t ifindex, int hwaddrlen, u_char *hwaddr, HSL_BOOL
send_notification)
```

Input Parameters

`ifindex`	Interface index.
`hwaddrlen`	Hardware length.
`hwaddrlen`	Hardware address.

`send_notification`

Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_set_arp_ageing_timeout

This function sets the ARP ageing time out value for an interface.

Syntax

```
int
hsl_ifmgr_set_arp_ageing_timeout (hsl_ifIndex_t ifindex, int arp_ageing_timeout)
```

Input Parameters

`ifindex` Interface index.

`arp_ageing_timeout`

ARP ageing time out value.

Output Parameters

None

Return Values

hsl_ifmgr_get_if_counters

This function gets an interface MAC counter.

Syntax

```
int
hsl_ifmgr_get_if_counters (hsl_ifIndex_t ifindex, struct hal_if_counters *cntrs)
```

Input Parameters

`ifindex` Interface index.

`cntrs` MAC counters.

Output Parameters

None

Return Values

hsl_ifmgr_collect_if_stat

This function processes interface statistics collection.

Syntax

```
void
hsl_ifmgr_collect_if_stat(void)
```

Input Parameters

None

Output Parameters

None

Return Values

None

IPv4 API Functions

The following table and subsection includes the IPv4 API functions.

Table 15: IPv5 API functions

Functions	Description
hsl_ifmgr_ipv4_address_add	Adds an IPV4 interface address.
hsl_ifmgr_ipv4_address_delete	Deletes an IPV4 interface address.

hsl_ifmgr_ipv4_address_add

This function adds an IPV4 interface address.

Syntax

```
int
hsl_ifmgr_ipv4_address_add (char *name, hsl_ifIndex_t ifindex, hsl_prefix_t *prefix,
                             u_char flags)
```

Input Parameters

name	Interface name.
ifindex	Interface index.
flags	

Output Parameters

prefix

Return Values

hsl_ifmgr_ipv4_address_delete

This function deletes an IPV4 interface address.

Syntax

```
int
hsl_ifmgr_ipv4_address_delete (char *name, hsl_ifIndex_t ifindex, hsl_prefix_t *prefix)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.

Output Parameters

`prefix`

Return Values

IPv6 API Functions

The following table and subsection includes the IPv4 API functions.

Table 16: IPv6 API functions

Functions	Description
hsl_ifmgr_ipv6_address_add	Adds an IPV6 interface address.
hsl_ifmgr_ipv6_address_delete	Deletes an IPV6 interface address.

hsl_ifmgr_ipv6_address_add

This function adds an IPV6 interface address.

Syntax

```
int
hsl_ifmgr_ipv6_address_add (char *name, hsl_ifIndex_t ifindex, hsl_prefix_t *prefix,
                            u_char flags)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.
`flags`	

Output Parameters

`prefix`

Return Values

hsl_ifmgr_ipv6_address_delete

This function deletes an IPV6 interface address.

Syntax

```
int
hsl_ifmgr_ipv6_address_delete (char *name, hsl_ifIndex_t ifindex, hsl_prefix_t *prefix)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.

Output Parameters

`prefix`

Return Values

Layer 2 API Functions

The following table and subsection includes the layer 2 API functions for HSL.

Table 17: Layer 2 API functions

Functions	Description
hsl_ifmgr_get_first_L2_port	Finds the first layer 2 port.
hsl_ifmgr_get_L2_parent	Finds any aggregated layer 2 port from a member layer 2 port.
hsl_ifmgr_L2_ethernet_create	Creates a layer 2 Ethernet interface.
hsl_ifmgr_L2_ethernet_register	Registers a layer 2 port.
hsl_ifmgr_L2_ethernet_delete	Deletes a layer 2 Ethernet interface.
hsl_ifmgr_L2_ethernet_delete2	Deletes a layer 2 Ethernet interface.
hsl_ifmgr_L2_ethernet_unregister	Unregisters a layer 2 port.
hsl_ifmgr_L2_link_down	Sets a link down for a level 2 interface.
hsl_ifmgr_L2_link_up	Sets a link up for a level 2 interface.
hsl_ifmgr_set_router_port	Sets a link down for a level 2 interface.
hsl_ifmgr_set_switch_port	Sets a port to a switch port.

hsl_ifmgr_get_first_L2_port

This function finds the first layer 2 port.

Syntax

```
struct hsl_if *
hsl_ifmgr_get_first_L2_port (struct hsl_if *ifp)
```

Input Parameters

`ifp`	Interface port.

Output Parameters

None

Return Values

hsl_ifmgr_get_L2_parent

This function finds any aggregated layer 2 port from a member layer 2 port.

Syntax

```
struct hsl_if *
hsl_ifmgr_get_L2_parent (struct hsl_if *ifp)
```

Input Parameters

`ifp`	Interface port.

Output Parameters

None

Return Values

hsl_ifmgr_L2_ethernet_create

This function creates a layer 2 Ethernet interface.

Syntax

```
int
hsl_ifmgr_L2_ethernet_create (char *name, hsl_mac_address_t mac,
                              u_int16_t mtu, u_int32_t speed,
                              u_int32_t duplex, u_int32_t flags,
                              void *sys_info,
                              int send_notification,
                              struct hsl_if **ppifp)
```

Input Parameters

`name`	Interface name.
`mac`	MAC address
`mtu`	Maximum transmission unit
`speed`	
`duplex`	Duplex mode
`flags`	

send_notification	
	Indicates whether or not to send a notification to protocol modules.
ppifp	

Output Parameters

sys_info	System information

Return Values

hsl_ifmgr_L2_ethernet_register

This function is a layer 2 port registration function.

Syntax

```
int
hsl_ifmgr_L2_ethernet_register (char *name, hsl_mac_address_t mac,
                                u_int16_t mtu, u_int32_t speed,
                                u_int32_t duplex, u_int32_t flags,
                                void *sys_info,
                                struct hsl_if **ppifp)
```

Input Parameters

name	Interface name.
mac	MAC address
mtu	Maximum transmission unit
speed	
duplex	Duplex mode
flags	
ppifp	

Output Parameters

sys_info	System information

Return Values

hsl_ifmgr_L2_ethernet_delete

This function deletes a layer 2 port, with the option of calling the notifier.

Syntax

```
void
hsl_ifmgr_L2_ethernet_delete (struct hsl_if *ifp,
                              int send_notification)
```

Input Parameters

`ifp`	Interface pointer.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

None

hsl_ifmgr_L2_ethernet_delete2

This function deletes a layer 2 port.

Syntax

```
void
hsl_ifmgr_L2_ethernet_delete2 (struct hsl_if *ifp,
                               int send_notification)
```

Input Parameters

`ifp`	Interface pointer.
`send_notification`	Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

None

hsl_ifmgr_L2_ethernet_unregister

This function is a layer 2 port unregistration function for an interface.

Syntax

```
int
hsl_ifmgr_L2_ethernet_unregister (struct hsl_if *ifp)
```

Input Parameters

`ifp`	Interface pointer.

Output Parameters

None

Return Values

None

hsl_ifmgr_L2_link_down

This function sets a link down for a level 2 interface. This function is called only from the link scan.

Syntax

```
int
hsl_ifmgr_L2_link_down (struct hsl_if *ifp, u_int32_t speed, u_int32_t duplex)
```

Input Parameters

`ifp`	Interface pointer.
`speed`	
`duplex`	

Output Parameters

None

Return Values

hsl_ifmgr_L2_link_up

This function sets a link up for a level 2 interface. This function is called only from the link scan.

Syntax

```
int
hsl_ifmgr_L2_link_up (struct hsl_if *ifp, u_int32_t speed, u_int32_t duplex)
```

Input Parameters

`ifp`	Interface pointer.
`speed`	
`duplex`	

Output Parameters

None

Return Values

hsl_ifmgr_set_router_port

This function sets a port to a router port. The input parameters has to be the layer 2 port.

Syntax

```
int
hsl_ifmgr_set_router_port (struct hsl_if *ifp, void *data, struct hsl_if **ppifp,
HSL_BOOL send_notification)
```

Input Parameters

`ifp`	Interface pointer.
`data`	
`ppifp`	
`send_notification`	ARP ageing time out value.

Output Parameters

None

Return Values

hsl_ifmgr_set_switch_port

This function sets a port to a switch port. The input port has to be an IP interface.

Syntax

```
int
hsl_ifmgr_set_switch_port (struct hsl_if *ifp, struct hsl_if **ppifp, HSL_BOOL
send_notification)
```

Input Parameters

`ifp`	Interface pointer.
`ppifp`	
`send_notification`	ARP ageing time out value.

Output Parameters

None

Return Values

Layer 3 API Functions

The following table and subsection includes the layer 3 API functions for HSL.

Table 18: Layer 3 API functions

Functions	Description
hsl_ifmgr_get_additional_L3_port	Finds the layer 3 port for a layer 2 port that does not have a matching VLAN.
hsl_ifmgr_get_matching_L3_port	Discovers the layer 3 port for a layer 2 port that has a matching VLAN.
hsl_ifmgr_L3_create	Creates an layer 3 interface.
hsl_ifmgr_L3_register	Registers a layer 3 interface.
hsl_ifmgr_L3_delete2	Deletes a layer 3 interface.
hsl_ifmgr_L3_delete	Deletes a layer 3 interface.
hsl_ifmgr_L3_unregister	Unregisters a layer 3 interface.
hsl_ifmgr_L3_loopback_register	Registers a layer 3 interface.
hsl_ifmgr_L3_cpu_if_register	Registers a layer 3 CPU interface.

hsl_ifmgr_get_additional_L3_port

This function finds the top layer 3 port for a layer 2 port that does not have a matching VLAN.

Syntax

```
int
hsl_ifmgr_get_additional_L3_port (struct hsl_if *ifp, hsl_vid_t vid)
```

Input Parameters

`ifp`	Interface port.
`vid`	VLAN identifier

Output Parameters

None

Return Values

hsl_ifmgr_get_matching_L3_port

This function discovers the top layer 3 ports for layer 2 port that have matching VLANs. For router ports, the VLAN is ignored. For SVIs, the matching interface based on VLAN is returned. This function increments the reference count. The caller must decrement the reference count after using the ifp returned.

Syntax

```
struct hsl_if *
hsl_ifmgr_get_matching_L3_port (struct hsl_if *ifp, hsl_vid_t vid)
```

Input Parameters

`ifp`	Interface port.

`vid` VLAN identifier

Output Parameters

None

Return Values

hsl_ifmgr_L3_create

This function creates an layer 3 interface.

Syntax

```
int
hsl_ifmgr_L3_create (char *ifname, u_char *hwaddr, int hwaddrlen,
                     int send_notification, void *data, struct hsl_if **ppifp)
```

Input Parameters

`ifname` Interface name

`hwaddr` Hardware address

`hwaddrlen` Hardware address length

`send_notification`

Indicates whether or not to send a notification to protocol modules.

`data`

`ppifp`

Output Parameters

None

Return Values

hsl_ifmgr_L3_register

This function registers a layer 3 interface.

Syntax

```
int
hsl_ifmgr_L3_register (char *ifname, u_char *hwaddr, int hwaddrlen,
                       void *data, struct hsl_if **ppifp)
```

Input Parameters

`ifname` Interface name

`hwaddr` Hardware address

`hwaddrlen` Hardware address length

`data`

```
      ppifp
```

Output Parameters

None

Return Values

hsl_ifmgr_L3_delete2

This function deletes a layer 3 interface.

Syntax

```
void
hsl_ifmgr_L3_delete2 (struct hsl_if *ifp,
                      int send_notification)
```

Input Parameters

`ifp` Interface pointer.

`send_notification`

Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

hsl_ifmgr_L3_delete

This function deletes a layer 3 interface, along with a notification option.

Syntax

```
void
hsl_ifmgr_L3_delete (struct hsl_if *ifp,
                     int send_notification)
```

Input Parameters

`ifp` Interface pointer.

`send_notification`

Indicates whether or not to send a notification to protocol modules.

Output Parameters

None

Return Values

None

hsl_ifmgr_L3_unregister

This function unregisters a layer 3 interface.

Syntax

```
int
hsl_ifmgr_L3_unregister (char *name, hsl_ifIndex_t ifindex)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.

Output Parameters

None

Return Values

hsl_ifmgr_L3_loopback_register

This function registers a layer 3 interface.

Syntax

```
int
hsl_ifmgr_L3_loopback_register (char *name, hsl_ifIndex_t ifindex, int mtu,
                                u_int32_t flags, void *osifp)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.
`mtu`	Maximum transmission unit.
`flags`	
`osifp`	

Output Parameters

None

Return Values

hsl_ifmgr_L3_cpu_if_register

This function registers a layer 3 CPU interface.

Syntax

```
int
hsl_ifmgr_L3_cpu_if_register (char *name, hsl_ifIndex_t ifindex, int mtu, int speed,
                              u_int32_t flags, char *hw_addr, void *osifp)
```

Input Parameters

`name`	Interface name.
`ifindex`	Interface index.
`mtu`	Maximum transmission unit.
`speed`	
`flags`	
`hw_addr`	Hardware address.
`osifp`	

Output Parameters

None

Return Values

Port Mirroring API Functions

The following subsection includes the port mirroring API functions.

Table 19: Port mirroring API functions

Functions	Description
hsl_ifmgr_set_portmirror	Sets port mirroring.
hsl_ifmgr_unset_portmirror	Unsets port mirroring.

hsl_ifmgr_set_portmirror

This function sets port mirroring.

Syntax

```
int
hsl_ifmgr_set_portmirror (hsl_ifIndex_t to_ifindex, hsl_ifIndex_t from_ifindex,
                          enum hal_port_mirror_direction direction)
```

Input Parameters

`to_ifindex`	Mirrored to the port.
`from_ifindex`	Mirrored from the port.
`direction`	Direction to set for mirroring.

Output Parameters

None

Return Values

hsl_ifmgr_unset_portmirror

This function unsets port mirroring.

Syntax

```
int
hsl_ifmgr_unset_portmirror (hsl_ifIndex_t to_ifindex, hsl_ifIndex_t from_ifindex,
                            enum hal_port_mirror_direction direction)
```

Input Parameters

`to_ifindex`	Mirrored to the port.
`from_ifindex`	Mirrored from the port.
`direction`	Direction to set for mirroring.

Output Parameters

None

CHAPTER 4 Platform Abstraction Layer

This chapter provides information about the Platform Abstraction Layer (PAL). It includes an overview, list of features and all of the relevant ACL functions.

Overview

ZebOS-XP provides platform independence by utilizing the Platform Abstraction Layer (PAL) collections of API functions. For each supported platform, these functions are implemented to use the system services of each platform to isolate ZebOS-XP daemons from the operating system.

PAL allows ZebOS-XP to support the same number of operating systems and management protocols. Moreover, the complexity of each component is handled by a PAL module for each OS. This allows a ZebOS-XP component to make just one call, and then the compile time options link in the appropriate PAL module fulfills each call. ZebOS-XP comes with a set of production-quality, fully-tested PAL module for the most common operating systems. Any porting efforts to non-supported platforms can be modeled on these PAL modules.

Features

The following subsection describes the features of PAL.

Memory Management

A memory management in PAL gives developers more control over memory, including setting the class and size of the requested storage.

Sockets

Sockets services are used to create TCP sockets. PAL sockets handle all socket functionality. For example, creation and deletion functionality, read and write functionality; and synchronization functionality. In addition, there are extensions to the socket calls that handle layer 2 and raw sockets.

String

The String abstraction includes string operations, including copying, scanning, parsing, and comparing strings and characters. It interacts with the memory abstraction to create or move memory cells.

STDLIB

The stdlib abstraction takes advantage of the most effective stdlib functions on the system. If there is an enhanced or particular functionality available for a feature, that functionality can be used in preference to the standard function.

Configuration Storage

PAL abstracts storage of configuration files. This allows platforms without a file system to store configuration information in any available memory model, including linear memory or flash memory.

Logging

Logging facilities are abstracted so that the system can log to any available facility, for example syslog, file, or console. These outputs can also run in parallel so that each output can be set independently to different log trap levels.

Kernel (Interfaces and Routes)

The PAL abstracts control and manage kernel routes and interfaces. This allows ZebOS-XP to not directly perceive how routes are manipulated.

Daemons

Process daemons are abstracted so that they are automatically handled on systems with this functionality.

Source

The source tree layout puts each protocol in its own directory. There is no platform-specific source in any of the directories. All calls, data types, and other components that are operating-system-specific or platform-specific are moved to the PAL code. All included modules either document other required included modules or automatically include the necessary modules.

Build

A “dummy” build environment helps determine if the protocols include any system functions. The dummy environment excludes the system header files, and each protocol that builds with the dummy environment does not include any system-dependent functions. This ensures that all modules run in any PAL supported environment, even if that environment does not support the STDLIB feature.

Module Compatibility

Changes, additional features, and enhancements to protocols are restricted to the source for that module. Changes made necessary by a platform are restricted to the PAL module for that platform. A PAL API can change to support an operating system feature. This ensures backward compatibility for a protocol modules with later versions of an API.

System Components

The following table lists PAL equivalents to system components:

Table 1:

System Component	PAL Equivalent
OS services	PAL_Services
Types	PAL_Types
Sockets	PAL_Sockets
Memory	PAL_Memory
String	PAL_String
Stdlib	PAL_Stdlib
Configuration/storage	PAL_Config
Interfaces	PAL_Interfaces
Logging	PAL_Log
Route	PAL_Route
Daemonization	PAL_Daemon

Data Structures

The following subsection list the data structures for PAL.

Common Data Structures

See the *Common Data Structures Developer Guide* for a description of these data structures used by multiple ZebOS-XP modules:

- `connected`
- `pal_in4_addr`
- `pal_in6_addr`
- `prefix`
- `rib`
- `stream`

pal_timeval

This data structure helps manage time value functions. It is defined in pal/dummy/pal_time.h.

Definition

```
struct pal_timeval {
  u_int32_t tv_sec;
  u_int32_t tv_usec;
};
```

pal_tzval

This data structure provides time zone information. It is defined in pal/dummy/pal_time.h.

Type	Definition
tz_minuteswest	How many minutes west of Greenwich Mean Time (GMT).
tz_dsttime	Nonzero if ever use Daylight Savings Time (DST).

Definition

```
struct pal_tzval {
  s_int32_t tz_minuteswest; /* how many minutes *west* of GMT */
  int       tz_dsttime;     /* nonzero if ever use DST */
};
```

pal_tm

This data structure includes the elements of time disassembled. It is defined in pal/dummy/pal_time.h.

Definition

```
struct pal_tm {
  u_int32_t tm_sec;
```

```
      u_int32_t tm_min;
      u_int32_t tm_hour;
      u_int32_t tm_mday;
      u_int32_t tm_mon;
      u_int32_t tm_year;
      u_int32_t tm_wday;
      u_int32_t tm_yday;
      u_int32_t tm_isdst;
    };
```

rib

This data structure helps manage RIB functions. It is defined in nsm/rib/rib.h.

Member	Description
next	Linked list
prev	Linked list
type	Type of this route
sub_type	Sub type of this route
distance	Distance
flags	Flags of this route
metric	Metric
uptime	Uptime
ext_flags	Extended flags of this route
client_id	NSM protocol provides four-octet client ID. To reduce memory consumption in RIB, this is defined as one octet. You can extend this member by changing this definition. The client_id is local to a system and therefore cannot be check pointed. But it is used for the graceful restart mechanism to mark the routes that are STALE based on client id. Therefore, for HA the client id will be the protocol id.
nexthop_num	Nexthop information
nexthop_active_num	Nexthop information
nexthop	Nexthop information
rmm_flags	RMM module flag
vrf	VRF pointer
kernel_ms_lnode	Kernel Msg Stagger Link-List node pointer
pid	Process ID
tag	Tag

Member	Description
pflags	Inform nexthop change
domain_conf	OSPF Domain info

Definition

```
struct rib
{
  /* Link list. */
  struct rib *next;
  struct rib *prev;
  /* Type of this route. */
  u_char type;
  /* Sub type of this route.  */
  u_char sub_type;
  /* Distance. */
  u_char distance;
  /* Flags of this route.  */
  u_char flags;
  /* Metric */
  u_int32_t metric;
  /* Uptime. */
  pal_time_t uptime;
  /* Extended flags of this route */
  u_char ext_flags;
#define RIB_EXT_FLAG_MROUTE                  0x01
#ifdef HAVE_HA
#define RIB_EXT_FLAG_HA_RIB_CHANGED          0x02
#define RIB_EXT_FLAG_HA_RIB_DELETED          0x04
#endif /* HAVE_HA */
#define RIB_EXT_FLAG_BLACKHOLE_RECURSIVE     0x08
  /* Client ID.  NSM protocol provide four octet client ID.  But to
     reduce memory consumption in RIB, this client_id is defined as
     one octet.  You can extend this restriction by changing this
     definition.  */
  /* XXX: Client_id is local to a system and therefore cannot be
   * checkpointed. But it is used for Graceful Restart mechanism to
   * mark the routes STALE based on client id.
   * Therefore, for HA the client id will be the protocol id. This will
   * be ensured by assigning the client_id as the protocol_id at time
   * of NSM client connect (in nsm_server_recv_service() ).
   */
  u_char client_id;
  /* Nexthop information. */
  u_char nexthop_num;
  u_char nexthop_active_num;
  struct nexthop *nexthop;
#ifdef HAVE_RMM
```

```
  /* RMM module flag.  */
  u_char rmm_flags;
#endif /* HAVE_RMM */
  /* VRF pointer.  */
  struct nsm_vrf *vrf;
#ifdef HAVE_STAGGER_KERNEL_MSGS
  /* Kernel Msg Stagger Link-List node pointer. */
  struct listnode *kernel_ms_lnode;
#endif /* HAVE_STAGGER_KERNEL_MSGS */
#ifdef HAVE_HA
  HA_CDR_REF nsm_rib_cdr_ref;
#endif /* HAVE_HA */
 /*Process ID */
 u_int32_t pid;
 /* Tag */
 u_int32_t tag;
 /* inform nexthop change */
 u_int32_t pflags;
#define NSM_ROUTE_CHG_INFORM_BGP (1 << 0)
#ifdef HAVE_BFD
#define NSM_ROUTE_CHG_BFD (1 << 1)
#define NSM_BFD_CONFIG_CHG (1 << 2)
#endif /* HAVE_BFD */
#ifdef HAVE_MPLS
#define NSM_ROUTE_HAVE_IGP_SHORTCUT (1 << 3)
#endif/* HAVE_MPLS */
#ifdef HAVE_VRF
 /*OSPF Domain info */
 struct nsm_ospf_domain_conf *domain_conf;
#endif /*HAVE_VRF*/
};#endif /* HAVE_RMM */
  struct nsm_mass_event *mass_event;
  /* VRF pointer.  */
  struct nsm_vrf *vrf;
#ifdef HAVE_STAGGER_KERNEL_MSGS
  /* Kernel Msg Stagger Link-List node pointer. */
  struct listnode *kernel_ms_lnode;
#endif /* HAVE_STAGGER_KERNEL_MSGS */
#ifdef HAVE_HA
  HA_CDR_REF nsm_rib_cdr_ref;
#endif /* HAVE_HA */
 /*Process ID */
 u_int32_t pid;
 /* inform nexthop change */
 u_int32_t pflags;
#define NSM_ROUTE_CHG_INFORM_BGP (1 << 0)
#ifdef HAVE_VRF
 /*OSPF Domain info */
 struct nsm_ospf_domain_conf *domain_conf;
#endif /*HAVE_VRF*/
```

```
};
```

PAL API Functions

The following subsection describes the PAL API functions. The following includes the API functions included in this subsection.

Functions	Description
pal_if_mip6_home_agent_set	Sets the home agent interface
pal_if_mip6_home_agent_unset	Unsets the home agent interface
pal_kernel_fib_create	Creates a FIB
pal_kernel_fib_delete	Deletes a FIB
pal_kernel_gratuitous_arp_send	Sends a gratuitous ARP message
pal_kernel_if_bind_vrf	Binds an interface to a virtual router
pal_kernel_if_flags_get	Gets the flags for an interface and writes the current value to the flags in the interface structure
pal_kernel_if_flags_set	Sets an interface flag and updates the actual interface so it is consistent
pal_kernel_if_flags_unset	Unsets an interface flag and updates the actual interface so it is consistent
pal_kernel_if_get_bw	Gets the bandwidth and writes the value to the interface
pal_kernel_if_get_hwaddr	Gets the hardware address
pal_kernel_if_get_index	Gets the interface index for the given interface
pal_kernel_if_get_metric	Get an interface’s metric
pal_kernel_if_get_mtu	Gets the inerface’s maximum transmission unit
pal_kernel_if_ipv4_address_add	Sets an IPv4 address, mask, and broadcast address for an interface
pal_kernel_if_ipv4_address_delete	Removes an IPv4 address, mask, and broadcast address from an interface
pal_kernel_if_ipv4_address_delete_all	Removes all IPv4 addresses from an interface
pal_kernel_if_ipv4_address_secondary_add	Sets an IPv4 secondary address, mask, and broadcast address for an interface
pal_kernel_if_ipv4_address_secondary_delete	Removes an IPv4 secondary address, mask, and broadcast address for an interface
pal_kernel_if_ipv4_address_update	Sets an IPv4 secondary address, mask, and broadcast address for an interface
pal_kernel_if_ipv6_address_add	Sets an IPv6 address, mask, and broadcast address for an interface
pal_kernel_if_ipv6_address_delete	Removes an IPv6 address, mask, and broadcast address for an interface

Functions	Description
pal_kernel_if_scan	Scans the kernel interface list and creates interfaces in the interface list
pal_kernel_if_unbind_vrf	Unbinds an interface from a virtual router
pal_kernel_if_update	Scans the kernel interface list and update interfaces
pal_kernel_ipv4_add	Add an entry to the kernel IPv4 forwarding table
pal_kernel_ipv4_del	Removes an entry in the kernel IPv4 forwarding table
pal_kernel_ipv4_forwarding_get	Gets the state of IPv4 forwarding in the kernel
pal_kernel_ipv4_forwarding_set	Sets the state of IPv4 forwarding in the kernel
pal_kernel_ipv4_update	Updates an entry in the kernel IPv4 forwarding table
pal_kernel_ipv6_add	Add an entry to the kernel IPv6 forwarding table
pal_kernel_ipv6_del	Removes an entry in the kernel IPv6 forwarding table
pal_kernel_ipv6_forwarding_get	Gets the state of IPv6 forwarding in the kernel
pal_kernel_ipv6_forwarding_set	Sets the state of IPv6 forwarding in the kernel
pal_kernel_ipv6_old_del	Removes an entry from the kernel IPv6 forwarding table
pal_kernel_ipv6_update	Updates an entry in the kernel IPv6 forwarding table
pal_kernel_L2_ipv4_resolve	Resolves an IPv4 address into a Layer 2 address
pal_kernel_route_scan	Scans the kernel routing table and loads the routes into the RIB
pal_kernel_start	Starts the kernel control manager
pal_kernel_stop	Stops the kernel control manager
pal_kernel_virtual_ipv4_add	Adds a virtual IP address to the given interface
pal_kernel_virtual_ipv4_delete	Deletes a virtual IP address from the given interface
pal_kernel_virtual_mac_add	Adds a virtual MAC address to the given interface
pal_kernel_virtual_mac_delete	Deletes a virtual MAC address from the given interface
pal_kernel_vrrp_start	Initializes the platform data for VRRP
pal_log_close	Closes a log message
pal_log_open	Opens a log message
pal_log_output	Outputs a log message
pal_log_start	Starts a log message
pal_log_stop	Stops a log message
pal_time_calendar	Converts the calendar time into string form

Functions	Description
pal_time_clock	Gets the number of clock ticks
pal_time_current	Gets the current time.
pal_time_gmt	Converts the local time to GMT form
pal_time_loc	Converts the local time to expanded form
pal_time_mk	Compresses the expanded struct tm to time_t
pal_time_start	Starts the time manager
pal_time_stop	Stops the time manager
pal_time_strf	Converts the expanded time to string form
pal_time_tzcurrent	Gets time of day and time zone information

pal_if_mip6_home_agent_set

This function sets the home agent interface.

API Call

```
result_t pal_if_mip6_home_agent_set (struct interface *ifp);
```

Input Parameters

ifp	A pointer to the interface

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_if_mip6_home_agent_unset

This function unsets the home agent interface.

API Call

```
result_t pal_if_mip6_home_agent_unset (struct interface *ifp);
```

Input Parameters

ifp	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_fib_create

This function creates a FIB in the forwarding plane for the given FIB identifier.

API Call

```
result_t pal_kernel_fib_create (fib_id_t fib_id);
```

Input Parameters

`fib_id`	FIB ID

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_fib_delete

This function deletes an FIB in the forwarding plane for the given FIB identifier.

API Call

```
result_t pal_kernel_fib_delete (fib_id_t fib_id);
```

Input Parameters

`fib_id`	FIB ID

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_gratuitous_arp_send

This function sends a gratuitous ARP message to the given interface.

API Call

```
result_t pal_kernel_gratuitous_arp_send (struct lib_globals *lib_node,
struct stream *ap, struct interface *ifp);
```

Input Parameters

`lib_node`	Global variables
`ap`	Gratuitous ARP message
`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_bind_vrf

This function binds an interface to a Virtual Router (VR) in the dataplane.

API Call

```
result_t pal_kernel_if_bind_vrf (struct interface *ifp, fib_id_t fib_id);
```

Input Parameters

`ifp`	A pointer to the interface
`fib_id`	VR context ID.

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_flags_get

This function gets the flags for an interface and writes the current value to the flags in the interface structure.

API Call

```
result_t pal_kernel_if_flags_get (struct interface *ifp);
```

Input Parameters

`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_flags_set

This function sets an interface flag and update the actual interface so it is consistent. This function uses the bit flag bit positions given by the PAL implementation.

API Call

```
result_t pal_kernel_if_flags_set (struct interface *ifp, u_int32_t flag);
```

Input Parameters

`ifp`	A pointer to the interface
`flag`	Flag

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_flags_unset

This function unsets an interface flag and update the actual interface so it is consistent. This function uses the bit flag positions given by the PAL implementation.

API Call

```
result_t pal_kernel_if_flags_unset (struct interface *ifp, u_int32_t flag);
```

Input Parameters

`ifp`	A pointer to the interface
`flag`	Flag

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_get_bw

This function gets the bandwidth and write the value to the interface.

API Call

```
result_t pal_kernel_if_get_bw (struct interface *ifp);
```

Input Parameters

ifp	A pointer to the interface

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_get_hwaddr

This function gets the hardware address.

API Call

```
result_t pal_kernel_if_get_hwaddr (struct interface *ifp);
```

Input Parameters

ifp	A pointer to the interface

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_get_index

This function gets the interface index for the given interface.

API Call

```
result_t pal_kernel_if_get_index (struct interface *ifp);
```

Input Parameters

ifp	A pointer to the interface

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_get_metric

This function gets an interface's metric.

API Call

```
result_t pal_kernel_if_get_metric (struct interface *ifp);
```

Input Parameters

`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_get_mtu

This function gets the interface's Maximum Transmission Unit (MTU).

API Call

```
result_t pal_kernel_if_get_mtu (struct interface *ifp);
```

Input Parameters

`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_add

This function sets an IPv4 address, mask, and broadcast address for an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_add (struct interface *ifp, struct connected *ifc);
```

Input Parameters

`ifp`	A pointer to the interface
`ifc`	A pointer to the connected address

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_delete

This function removes an IPv4 address, mask, and broadcast address from an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_delete (struct interface *ifp,
                                             struct connected *ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the connected address

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_delete_all

This function removes all IPv4 addresses from an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_delete_all (struct interface *ifp,struct connected
*ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the top of connected addresses

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_secondary_add

This function sets an IPv4 secondary address, mask, and broadcast address for an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_secondary_add (struct interface *ifp, struct
connected *ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the connected address

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_secondary_delete

This function removes an IPv4 secondary address, mask, and broadcast address from an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_secondary_delete (struct interface *ifp, struct
connected *ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the connected address

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv4_address_update

This function updates the primary IPv4 address for an interface.

API Call

```
result_t pal_kernel_if_ipv4_address_update (struct interface *ifp, struct connected
*ifc_old, struct connected *ifc_new);
```

Input Parameters

ifp	A pointer to the interface
ifc_old	A pointer to the connected address to delete
ifc_new	A pointer to the connected address to add

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv6_address_add

This function sets an IPv6 address, mask, and broadcast address for an interface.

API Call

```
result_t pal_kernel_if_ipv6_address_add (struct interface *ifp, struct connected *ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the connected address

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

Some other value when the function fails

pal_kernel_if_ipv6_address_delete

This function removes an IPv6 address, mask, and broadcast address from an interface.

API Call

```
result_t pal_kernel_if_ipv6_address_delete (struct interface *ifp, struct connected
*ifc);
```

Input Parameters

ifp	A pointer to the interface
ifc	A pointer to the connected address

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_scan

This function scans the kernel interface list and create interfaces in the interface list.

API Call

```
result_t pal_kernel_if_scan (void);
```

Input Parameters

None

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_unbind_vrf

This function unbinds an interface from a virtual router in the dataplane.

API Call

```
result_t
pal_kernel_if_unbind_vrf (struct interface *ifp, fib_id_t table)
```

Input Parameters

`ifp`	A pointer to the interface
`fib_id`	VR context ID

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_if_update

This function scans the kernel interface list and update interfaces.

API Call

```
void pal_kernel_if_update (void);
```

Input Parameters

None

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv4_add

This function adds an entry to the kernel IPv4 forwarding table.

API Call

```
result_t pal_kernel_ipv4_add (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv4_del

This function removes an entry from the kernel IPv4 forwarding table.

API call

```
result_t pal_kernel_ipv4_del (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv4_forwarding_get

This function gets the state of IPv4 forwarding in the kernel.

API Call

```
result_t pal_kernel_ipv4_forwarding_get (s_int32_t * state);
```

Input Parameters

None

Output Parameters

`state`	A pointer to the state (non-zero for on, zero for off)

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv4_forwarding_set

This function sets the IPv4 forwarding state in the kernel.

API Call

```
result_t pal_kernel_ipv4_forwarding_set (s_int32_t state);
```

Input Parameters

`state`	New state (non-zero = on)

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv4_update

This function updates an entry in the kernel IPv4 forwarding table.

API Call

```
result_t pal_kernel_ipv4_update (struct prefix *p, struct rib *r, struct rib *s);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the current RIB entry
`s`	A pointer to the new RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv6_add

This function adds an entry to the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_add (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv6_del

This function removes an entry from the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_del (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv6_forwarding_get

This function gets the state of IPv6 forwarding in the kernel.

API Call

```
result_t pal_kernel_ipv6_forwarding_get (s_int32_t * state);
```

Input Parameters

None

Output Parameters

`state`	A pointer to the state

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv6_forwarding_set

This function sets the state of IPv6 forwarding in the kernel.

API Call

```
result_t pal_kernel_ipv6_forwarding_set (s_int32_t state);
```

Input Parameters

`state`	New state (non-zero = on)

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_ipv6_old_del

This function removes an entry from the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_old_del (struct prefix_ipv6 *dest,
                                   struct pal_in6_addr *gate,
                                   u_int32_t index,
                                   u_int32_t flags, u_int32_t table);
```

Input Parameters

`dest`	Destination prefix
`gate`	Gateway address

`index`	Interface index
`flags`	Route flags
`table`	Table ID

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

pal_kernel_ipv6_update

This function updates an entry in the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_update (struct prefix *p, struct rib *r, struct rib *s);
```

Input Parameters

`p`	A pointer to the prefix
`r`	A pointer to the current RIB entry
`s`	A pointer to the new RIB entry

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_L2_ipv4_resolve

This function resolves an IPv4 address into a Layer 2 address.

API Call

```
result_t pal_kernel_l2_ipv4_resolve (u_int32_t instance, u_int32_t ip_addr,
u_int8_t * l2_addr);
```

Input Parameters

`instance`	The instance
`ip_addr`	The IPv4 address to resolve

Output Parameters

`l2_addr`	A pointer to the place to put the Layer 2 (L2) address.

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_route_scan

This function scans the kernel routing table and loads the routes into the RIB.

API Call

```
result_t pal_kernel_route_scan ();
```

Input Parameters

None

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_start

This function starts the kernel control manager. This sets up any needed variables, hooks into the OS, and prepares the kernel for transactions, as appropriate. It is only called during startup. The handle returned is stored in the library globals. If this is called multiple times without an intervening stop, it must return the same handle.

API Call

```
result_t pal_kernel_start (void);
```

Input Parameters

None

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_stop

This function stops the kernel control manager. This finishes any pending transactions, and shuts down the kernel control manager, breaking any previously created connections to the kernel or OS. It also frees any resources allocated by the kernel control manager. It is only called during the shutdown process. The stops and starts must be balanced, so stop must be called the same number of times as start before the stop is committed.

API Call

```
result_t pal_kernel_stop (void);
```

Input Parameters

None

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_virtual_ipv4_add

This function adds a virtual IP address to the given interface.

API Call

```
result_t pal_kernel_virtual_ipv4_add (struct lib_globals *lib_node, struct pal_in4_addr
*vip, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vip`	Virtual IP address
`ifp`	A pointer to the interface
`owner`	Owner status of this address
`vrid`	VRRP Virutal Router ID.

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_virtual_ipv4_delete

This function deletes a virtual IP address from the given interface.

API Call

```
result_t pal_kernel_virtual_ipv4_delete (struct lib_globals *lib_node, struct
pal_in4_addr *vip, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vip`	Virtual IP address
`ifp`	A pointer to the interface
`owner`	Owner status of this address

`vrid`	VRRP Virtual Router ID

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_virtual_mac_add

This function adds a virtual MAC address to the given interface. This MAC address is specified in RFC 3678 as `00-00-5E-00-01-<VRID>`.

API Call

```
result_t pal_kernel_virtual_mac_add (struct lib_globals *lib_node, u_int8_t vrid,
ztruct interface *ifp);
```

Input Parameters

`lib_node`	Global variables
`vrid`	VRRP Virtual Router ID
`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_virtual_mac_delete

This function deletes a virtual MAC address from the given interface. This MAC address is specified in RFC 3678 as `00-00-5E-00-01-<VRID>`.

API Call

```
result_t pal_kernel_virtual_mac_delete (struct lib_globals *lib_node, u_int8_t vrid,
struct interface *ifp);
```

Input Parameters

`lib_node`	Global variables
`vrid`	VRRP Virtual Router ID
`ifp`	A pointer to the interface

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_kernel_vrrp_start

This function initializes the platform data for VRRP.

API Call

```
result_t pal_kernel_vrrp_start (struct lib_globals *lib_node);
```

Input Parameters

`lib_node`	Global variables

Output Parameters

None

Return Value

`RESULT_OK` when the function succeeds

Some other value when the function fails

pal_log_close

This function closes a log, and commits any outstanding buffered writes to it.

API Call

```
result_t pal_log_close (struct lib_globals *libnode, struct zlog *zl);
```

Input Parameters

`libnode`	The log file to use.
`zl`	The module ID for the log output.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_log_open

This function opens a log.

API Call

```
result_t pal_log_open (struct lib_globals *libnode, struct zlog *zl,
enum log_destination dest);
```

Input Parameters

libnode	The module instance for the log output.
zl	The log instance.
dest	The log destination.

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_log_output

This function outputs a log message to the debugging output device. This function writes the provided string to the log output, if the given priority of the entry is at the current logging priority or higher. The output might be timestamped, but this is done by routines called by this routine, instead of by the routine that called this routine.

API Call

```
result_t pal_log_output (struct lib_globals *zg, struct zlog *zl, char * priority_str,
char *protocol, char * message);
```

Input Parameters

zg	The log file to use.
zl	The log module.
priority_str	The priority of the message as a string.
protocol	The string representing the protocol or module.
message	The buffer containing the data to be logged.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_log_start

This function starts the logging output manager. It sets up needed variables and hooks into the OS, and prepares the logging device for transactions.

API Call

```
    result_t pal_log_start (struct lib_globals *lib_node);
```

Input Parameters

lib_node	The library globals.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_log_stop

This function stops logging, and finishes any pending transactions; shutting down the logging output manager, and breaking any previously created connections to the OS and output devices. It also frees any resources allocated by the logging output manager. It is only called during the shutdown process.

API Call

```
    result_t pal_log_stop (struct lib_globals *lib_node);
```

Input Parameters

log	The log file to use.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_calendar

This function converts the calendar time into string form. The calendar time is often obtained through a call to pal_time_current (); this function is used to replaced ctime ();

Syntax

```
result_t
pal_time_calendar (const pal_time_t *tp,
                        char *buf)
```

Input Parameters

tp	A pointer to the time_t to use

Output Parameters

buf	Pointer to character buffer

Return Values

RESULT_OK for success, else the error which occurred

pal_time_clock

This function gets the number of clock ticks since the system started; this call replaces clock ().

Syntax

```
pal_clock_t
pal_time_clock (void)
```

Input Parameters

None

Output Parameters

None

Return Values

pal_time_current

This function gets the current time. Returns the current time, plus sets the time_t at the end of the provided pointer (unless the pointer is NULL, then it only returns the current time). This replaces the time () call.

Syntax

```
pal_time_t
pal_time_current (pal_time_t *tp)
```

Input Parameters

None

Output Parameters

`tp`	A pointer to a time_t to set or NULL.

Return Values

pal_time_gmt

This function converts the local time to GMT (UTC) in expanded form.

Syntax

```
result_t
pal_time_gmt (pal_time_t *tp,
              struct pal_tm *gmt)
```

Input Parameters

`tp`	A pointer to the time to convert

Output Parameters

`*gmt`	A pointer to where to put the expanded GMT

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_loc

This function converts the local time to expanded form.

Syntax

```
result_t
pal_time_loc (pal_time_t *tp,
              struct pal_tm *loc)
```

Input Parameters

`tp`	A pointer to the time to convert.

Output Parameters

`*loc`	A pointer to where to put the expanded form.

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_mk

This function compresses the expanded struct tm to time_t.

Syntax

```
pal_time_t
pal_time_mk (struct pal_tm *tp)
```

Input Parameters

`tp`	A pointer to a struct tm to use.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_start

This function starts the time manager.

Syntax

```
pal_handle_t
pal_time_start (struct lib_globals *lib_node)
```

Input Parameters

`libnode`	The log file to use.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_stop

This function stops the time manager.

Syntax

```
result_t
pal_time_stop (struct lib_globals *lib_node)
```

Input Parameters

`libnode`	The log file to use.

Output Parameters

None

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_strf

This function converts the expanded time to string form.

Syntax

```
size_t
pal_time_strf (char *s,
               size_t smax,
               const char *fmt,
               const struct pal_tm *tp)
```

Input Parameters

`smax`	Maximum length of string
`*fmt`	A pointer to the format to use
`*tp`	A pointer to the struct tm to use

Output Parameters

`*s`	A pointer to where to put the string.

Return Values

RESULT_OK if function succeeds

Error occurs if function fails

pal_time_tzcurrent

This function gets time of day and time zone information. Puts the current time, plus the current time zone in the provided space. Does not return time if the "t" parameter is NULL; does not return time zone if the "tz" parameter is NULL.

Syntax

```
void
pal_time_tzcurrent (struct pal_timeval *t,
                    struct pal_tzval *tz)
```

Input Parameters

None

Output Parameters

`t`	Pointer to the timeval to use
`tz`	Pointer to the tzval to use

Return Values

None

CHAPTER 5 System Messages

This chapter describes the messages related to the Hardware Service Layer. All of these messages are sent from NSM.

HAL Initialization Messages

Message	This Message to...
HAL_MSG_INIT	Global hardware SDK initialization message.
HAL_MSG_DEINIT	Global hardware SDK deinitialization message.

Interface Manager Messages

Message	This Message is
HAL_MSG_IF_GETLINK	Get interface list. Retrieve a list of all interfaces known to Interface manager.
HAL_MSG_IF_GET_METRIC	Get interface metric.
HAL_MSG_IF_GET_MTU	Get interface max transmit unit.
HAL_MSG_IF_SET_MTU	Set interface max transmit unit.
HAL_MSG_IF_GET_DUPLEX	Get interface duplex.
HAL_MSG_IF_SET_DUPLEX	Set interface duplex.
HAL_MSG_IF_SET_AUTONEGO	Enable/Disable auto-negotiation on interface
HAL_MSG_IF_GET_HWADDR	Get interface hardware (Ethernet) address.
HAL_MSG_IF_SET_HWADDR	Set interface hardware (Ethernet) address.
HAL_MSG_IF_GET_FLAGS	Get interface flags (such as, admin up/down).
HAL_MSG_IF_SET_FLAGS	Set interface flags (such as, admin up/down).
HAL_MSG_IF_UNSET_FLAGS	Unset interface flags (such as, admin up/down).
HAL_MSG_IF_GET_BW	Get interface bandwidth.
HAL_MSG_IF_SET_BW	Set interface bandwidth (10/100/Giga/10 Gig).
HAL_MSG_IF_COUNTERS_GET	Extract interface counters (packet/octet statistics).

Message	This Message is
HAL_MSG_IF_GET_ARP_AGEING_TIMEOUT	Get ARP ageing timeout.
HAL_MSG_IF_SET_ARP_AGEING_TIMEOUT	Set ARP ageing timeout.
HAL_MSG_IF_L3_INIT	Initialize Layer-3 routing.
HAL_MSG_IF_IPV4_NEWADDR	Add interface address (IPv4 Layer-3).
HAL_MSG_IF_IPV4_DELADDR	Remove interface address (IPv4 Layer-3).
HAL_MSG_IF_IPV6_ADDRESS_ADD	Add interface address (IPv6 Layer-3).
HAL_MSG_IF_IPV6_ADDRESS_DELETE	Remove interface address (IPv6 Layer-3).
HAL_MSG_IF_CREATE_SVI	Create Shared VLAN (VLAN termination) Interface.
HAL_MSG_IF_DELETE_SVI	Create Shared VLAN (VLAN termination) Interface.
HAL_MSG_IF_SET_PORT_TYPE	Switch port between Layer-2 and Layer-3 mode.
HAL_MSG_IPV4_INIT	Initialize IPv4 Layer-3 routing.
HAL_MSG_IPV4_DEINIT	De-initialize IPv4 Layer-3 routing.
HAL_MSG_FIB_CREATE	Create forwarding database.
HAL_MSG_FIB_DELETE	Flush forwarding database.
HAL_MSG_IPV4_UC_ADD	Add IPv4 unicast route.
HAL_MSG_IPV4_UC_DELETE	Delete IPv4 unicast route.
HAL_MSG_IPV4_UC_UPDATE	Update IPv4 unicast route. (Next hop update)
HAL_MSG_IPV6_INIT	Initialize IPv6 Layer-3 routing.
HAL_MSG_IPV6_DEINIT	De-initialize IPv6 Layer-3 routing.
HAL_MSG_IPV6_UC_INIT	Initialize IPv6 unicast.
HAL_MSG_IPV6_UC_DEINIT	De-initialize IPv6 unicast.
HAL_MSG_IPV6_UC_ADD	Add IPv6 unicast route.
HAL_MSG_IPV6_UC_DELETE	Delete IPv6 unicast route.
HAL_MSG_IPV6_UC_UPDATE	Update IPv6 unicast route (Next hop update)

Layer-2 General Messages

Message	This Message is
HAL_MSG_BRIDGE_SET_PORT_STATE	Set port xSTP port state.

Bridge Messages

Message	This Message is
HAL_MSG_BRIDGE_INIT	Initialize Layer-2 Bridging.
HAL_MSG_BRIDGE_DEINIT	De-initialize Layer-2 Bridging.
HAL_MSG_BRIDGE_ADD	Add a Layer-2 bridge
HAL_MSG_BRIDGE_DELETE	Delete a Layer-2 bridge.
HAL_MSG_BRIDGE_SET_AGEING_TIME	Set Layer-2 FDB entry max age.
HAL_MSG_BRIDGE_SET_LEARNING	Enable/Disable learning on a bridge.
HAL_MSG_BRIDGE_ADD_PORT	Add port to a bridge.
HAL_MSG_BRIDGE_DELETE_PORT	Remove port from a bridge.

MSTP Messages

Message	This Message is
HAL_MSG_BRIDGE_ADD_INSTANCE	Add an MSTP bridge instance.
HAL_MSG_BRIDGE_DELETE_INSTANCE	Remove an MSTP bridge instance.
HAL_MSG_BRIDGE_ADD_VLAN_TO_INSTANCE	Add VLAN to an instance.
HAL_MSG_BRIDGE_DELETE_VLAN_FROM_INSTANCE	Remove VLAN from instance.

VLAN Messages

Message	This Message is
HAL_MSG_VLAN_INIT	Initialize VLAN database.
HAL_MSG_VLAN_DEINIT	De-initialize VLAN database.
HAL_MSG_VLAN_ADD	Add VLAN to a bridge.
HAL_MSG_VLAN_DELETE	Remove VLAN from a bridge.
HAL_MSG_VLAN_SET_PORT_TYPE	Set port type (tagged/untagged and ingress filter on/off).
HAL_MSG_VLAN_SET_DEFAULT_PVID	Set default VLAN ID on port.
HAL_MSG_VLAN_ADD_VID_TO_PORT	Add port to VLAN.
HAL_MSG_VLAN_DELETE_VID_FROM_PORT	Remove port from VLAN.
HAL_MSG_VLAN_CLASSIFIER_ADD	Add VLAN classifier.
HAL_MSG_VLAN_CLASSIFIER_DELETE	Remove VLAN classifier.
HAL_MSG_VLAN_STACKING_ENABLE	Enable VLAN stacking on port.
HAL_MSG_VLAN_STACKING_DISABLE	Disable VLAN stacking on port.

Flow Control Messages

Message	This Message is
HAL_MSG_FLOW_CONTROL_INIT	Initialize flow control.
HAL_MSG_FLOW_CONTROL_DEINIT	De-initialize flow control.
HAL_MSG_FLOW_CONTROL_SET	Enable/Disable flow control negotiation on the port.
HAL_MSG_FLOW_CONTROL_STATISTICS	Get flow control statistics for a port number of rxpause/txpause frames sent/received.

Rate Limiting Messages

Message	This Message is
HAL_MSG_RATELIMIT_INIT	Initialize Rate Limiting.
HAL_MSG_RATELIMIT_DEINIT	De-initialize Rate Limiting.

Message	This Message is
HAL_MSG_RATELIMIT_BCAST	Enable broadcast Rate Limiting.
HAL_MSG_RATELIMIT_BCAST_DISCARDS_GET	Get number of broadcast discards.
HAL_MSG_RATELIMIT_MCAST	Enable multicast Rate Limiting.
HAL_MSG_RATELIMIT_MCAST_DISCARDS_GET	Get number of multicast discards.
HAL_MSG_RATELIMIT_DLF_BCAST	Enable Unicast Unknown Destination rate limiting.
HAL_MSG_RATELIMIT_DLF_BCAST_DISCARDS_GET	Get number of Unicast Unknown Destination discards.

IGMP Snooping Messages

Message	This Message is
HAL_MSG_IGMP_SNOOPING_INIT	Initialize IGMP Snooping.
HAL_MSG_IGMP_SNOOPING_DEINIT	De-initialize IGMP Snooping.
HAL_MSG_IGMP_SNOOPING_ENABLE	Enable IGMP Snooping on a port.
HAL_MSG_IGMP_SNOOPING_DISABLE	Disable IGMP snooping on a port.

Layer-2 FDB Messages

Message	This Message is
HAL_MSG_L2_FDB_INIT	Initialize Layer-2 forwarding database.
HAL_MSG_L2_FDB_DEINIT	De-initialize Layer-2 forwarding database.
HAL_MSG_L2_FDB_ADD	Add Layer-2 forwarding entry.
HAL_MSG_L2_FDB_DELETE	Remove Layer-2 forwarding entry.
HAL_MSG_L2_FDB_UNICAST_GET	Get all unicast non static entries.
HAL_MSG_L2_FDB_MULTICAST_GET	Get all multicast non static entries.
HAL_MSG_L2_FDB_FLUSH_PORT	Flush FDB for specific port.

Port Mirroring Messages

Message	This Message is
`HAL_MSG_L2_PMIRROR_INIT`	Initialize mirroring.
`HAL_MSG_L2_PMIRROR_DEINIT`	De-initialize mirroring.
`HAL_MSG_L2_PMIRROR_SET`	Enable `tx/rx` mirroring of port to monitoring port.
`HAL_MSG_L2_PMIRROR_UNSET`	Disable `tx/rx` mirroring of port to monitoring port.

Link Aggregation Messages

Message	This Message is
`HAL_MSG_LACP_INIT`	Initialize link aggregation.
`HAL_MSG_LACP_DEINIT`	Deinit link aggregation.
`HAL_MSG_LACP_ADD_AGGREGATOR`	Add aggregator interface.
`HAL_MSG_LACP_DELETE_AGGREGATOR`	Remove aggregation interface.
`HAL_MSG_LACP_ATTACH_MUX_TO_AGGREGATOR`	Add port to aggregator.
`HAL_MSG_LACP_DETACH_MUX_FROM_AGGREGATOR`	Remove port from aggregator.
`HAL_MSG_LACP_PSC_SET`	Set outgoing port selection criteria.
`HAL_MSG_LACP_COLLECTING`	
`HAL_MSG_LACP_DISTRIBUTING`	
`HAL_MSG_LACP_COLLECTING_DISTRIBUTING`	

802.1x Messages

Message	This Message is
`HAL_MSG_8021x_INIT`	Initialize 802.1x authentication.
`HAL_MSG_8021x_DEINIT`	De-initialize 802.1x authentication.
`HAL_MSG_8021x_PORT_STATE`	Set port state (blocked/ blocked in/ authenticated).

Multicast Messages

Message	This Message is
HAL_MSG_IPV4_MC_INIT	IPv4 multicast init.
HAL_MSG_IPV4_MC_DEINIT	De-initialize IPv4 multicast.
HAL_MSG_IPV4_MC_PIM_INIT	Initialize PIM (v4).
HAL_MSG_IPV4_MC_PIM_DEINIT	De-initialize PIM (v4).
HAL_MSG_IPV4_MC_VIF_ADD	Add an IPv4 interface to a multicast router.
HAL_MSG_IPV4_MC_VIF_DEL	Remove an IPv4 interface from a multicast router.
HAL_MSG_IPV4_MC_MRT_ADD	Add an IPv4 multicast route.
HAL_MSG_IPV4_MC_MRT_DEL	Remove an IPv4 multicast route.
HAL_MSG_IPV4_MC_SG_STAT	Get multicast route usage statistics.
HAL_MSG_IPV6_MC_INIT	Initialize IPv6 multicast.
HAL_MSG_IPV6_MC_DEINIT	De-initialize IPv6 multicast.
HAL_MSG_IPV6_MC_PIM_INIT	Initialize PIM (v6).
HAL_MSG_IPV6_MC_PIM_DEINIT	De-initialize PIM (v6).
HAL_MSG_IPV6_MC_VIF_ADD	Add an IPv6 interface to a multicast router.
HAL_MSG_IPV6_MC_VIF_DEL	Remove an IPv6 interface from a multicast router.
HAL_MSG_IPV6_MC_MRT_ADD	Add an IPv6 multicast route.
HAL_MSG_IPV6_MC_MRT_DEL	Remove an IPv6 multicast route.
HAL_MSG_IPV6_MC_SG_STAT	Get multicast route IPv6 usage statistics.

ARP Messages

Message	This Message is
HAL_MSG_ARP_ADD	Add an ARP entry.
HAL_MSG_ARP_DEL	Delete an ARP entry.
HAL_MSG_ARP_CACHE_GET	Get all ARP entries.

- Takes care of cleaning up the data structures and protocol level information during the time of exit.

Data Structures for Layer 2 MRIB

The functions in this chapter refer to the data structures described in this section.

l2mrib_master

This data structure in `l2mrib.h` holds an L2 related information.

Type	Definition
ipi_vr	
lib_global	
l2mrib_mcast	
list	
list	

Definition

```
struct l2mrib_master
{
  struct ipi_vr *vr;

  struct lib_globals *zg;

  struct l2mrib_mcast *l2mcast;

  struct list *mcast_bridge_list;

  struct list *bridge_config; /* struct br_config*/

  u_char config_flag;
#define CONFIG_IGMP_SNOOP_DISABLED            (1<<0)
#define CONFIG_MLD_SNOOP_DISABLED             (1<<1)

#ifdef HAVE_DISABLE_IGMP_SNOOP
#define CONFIG_IGMP_SNOOP_ENABLED             (1<<2)
#endif /* HAVE_DISABLE_IGMP_SNOOP */

#ifdef HAVE_DISABLE_MLD_SNOOP
#define CONFIG_MLD_SNOOP_ENABLED              (1<<3)
#endif /* HAVE_DISABLE_MLD_SNOOP */
};
```

l2mrib_mcast

This data structure in `l2mrib.h` maintains the details of IGMP/MLD snooping such as IGMP/MLD instances, input/output buffer, svc registration ID any many more.

Type	Definition
2mrib_master	
stream	Packet Input/Output buffer
igmp_instance	IGMP instance
igmp_svc_reg_id	IGMP L2 Service Registration ID
mld_instance	MLD Instance
mld_svc_reg_id	MLD L2 Service Registration ID

Definition

```
struct l2mrib_mcast
{
  struct l2mrib_master *l2mm;

  /* Packet Input/Output buffer */
  struct stream *iobuf;

  /* Packet Output buffer */
  struct stream *obuf;

#ifdef HAVE_IGMP_SNOOP
  /* IGMP Instance */
  struct igmp_instance *igmp_inst;

  /* IGMP L2 Service Registration ID */
  void *igmp_svc_reg_id;
#endif

#ifdef HAVE_MLD_SNOOP
  /* MLD Instance */
  struct mld_instance *mld_inst;

  /* MLD L2 Service Registration ID */
  void *mld_svc_reg_id;
#endif

enum
  {
    L2MRIB_UNKNOWN_MCAST_FLOOD = 0,
    L2MRIB_UNKNOWN_MCAST_DISCARD = 1,
  }l2mrib_unknown_mcast;
```

l2mrib_bridge

This structure in `l2mrib.h` maintains all the bridge-related information, which it has processed from NSM, from the messages.

Member name	Description
bridge_name	
bridge_type	
is_enabled	
avl_tree	
thread	

Definition

```
struct l2mrib_bridge
{
  struct l2mrib_master *l2mm;

  u_int8_t      bridge_name[L2MRIB_BRIDGE_NAME_LEN+1];
  u_int8_t      bridge_type;

  u_int8_t      is_enabled;

  struct avl_tree *port_list;

  struct avl_tree *br_inst_list;

  struct avl_tree *vlan_table;

  struct avl_tree *snoop_entry;
  struct thread *t_snoop_entry_send;
};
```

l2mrib_if

This structure in `l2mrib.h` maintains all the bridge-related information, which it has processed, from messages from NSM.

Member name	Description
l2mrib_bridge	
l2mrib_port	
l2mrib_vlan	
ptree	
avl_tree	
l2mrib_vlan_bmp	

Definition

```
struct l2mrib_if
{
  struct l2mrib_bridge      *br;

  struct l2mrib_port        *l2port;

  struct l2mrib_vlan        *vlan;

#ifdef HAVE_IGMP_SNOOP
  struct ptree *igmpsnp_gmr_tib;
#endif

#ifdef HAVE_MLD_SNOOP
  struct ptree *mldsnp_gmr_tib;
#endif

  u_char if_state;
#define L2MRIB_IF_DEFAULT                       (1 << 0)
#define L2MRIB_IF_ENABLED                       (1 << 1)


struct l2mrib_port
{
  struct interface *ifp;

  struct avl_tree *port_inst_list;

  struct l2mrib_vlan_bmp staticMemberBmp;
};

struct l2mrib_port
{
  struct interface *ifp;
  struct avl_tree *port_inst_list;

  struct l2mrib_vlan_bmp staticMemberBmp;
};
```

l2mrib_port_instances

This structure in `l2mrib.h` maintains all port states

Member name	Description
instance_id	

Definition

```
struct l2mrib_port_instance
{
  u_int16_t instance_id;

  u_char state;
#define L2MRIB_PORT_STATE_DISABLED              1
#define L2MRIB_PORT_STATE_LISTENING             2
#define L2MRIB_PORT_STATE_LEARNING              3
#define L2MRIB_PORT_STATE_FORWARDING            4
#define L2MRIB_PORT_STATE_DISCARDING            5
};
```

l2mrib_bridge_instances

This structure in `l2mrib.h` maintains VLANS to MSTP/PVRST instance mapping

Member name	Description
instance_id	
avl_tree	

Definition

```
struct l2mrib_bridge_instance
{
  u_int16_t instance_id;
#ifdef HAVE_VLAN
  struct avl_tree *vlan_table;
#endif
};
```

Command API

The functions in this section are called by the commands in the *Multicast Routing Information Base Command Reference*.

Function	Description
hal_igmp_snooping_enable	Enables IGMP snooping for the bridge.
hal_igmp_snooping_disable	Disables IGMP snooping for the bridge
hal_igmp_snooping_if_enable	Enables IGMP snooping on an interface
hal_igmp_snooping_if_disable	Disables IGMP snooping on an interface
hal_igmp_snooping_add_entry	Adds a multicast entry for the source, group for a given VLAN
hal_igmp_snooping_delete_entry	Deletes a multicast entry for the source, group for a given VLAN
hal_mld_snooping_enable	Enables MLD snooping for the bridge
hal_mld_snooping_disable	Disables MLD snooping for the bridge
hal_mld_snooping_if_enable	Enables MLD snooping on an interface
hal_MLD_snooping_if_disable	Disables MLD snooping on an interface
hal_mld_snooping_add_entry	Adds a multicast entry for the source, group for a given VLAN
hal_mld_snooping_delete_entry	Deletes a multicast entry for the source, group for a given VLAN

Include File

Except where noted otherwise, you need to include `l2mrib.h` to call the functions in this section.

hal_igmp_snooping_enable

The API enables IGMP snooping for the bridge.

Syntax

```
int hal_igmp_snooping_enable (char *bridge_name);
```

Input Parameters

`bridge_name`	Bridge name

mOutput Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_ENABLE when the IGMP snooping for the bridge is not enabled

HAL_SUCCESS when the function succeeds

hal_igmp_snooping_disable

The API disables IGMP snooping for the bridge.

Syntax

```
int hal_igmp_snooping_disable (char *bridge_name);
```

Input Parameters

`bridge_name`	Bridge name

mOutput Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_DISABLE when the IGMP snooping for the bridge is not disabled

HAL_SUCCESS when the function succeeds

hal_igmp_snooping_if_enable

The API enables IGMP snooping on an interface.

Syntax

```
hal_igmp_snooping_if_enable (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index value

Output Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_ENABLE when the IGMP snooping on the interface is not enabled

HAL_SUCCESS when the function succeeds

hal_igmp_snooping_if_disable

The API disables IGMP snooping on an interface.

Syntax

```
hal_igmp_snooping_if_disable (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index value

Output Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_DISABLE when the IGMP snooping on the bridge is not disabled

HAL_SUCCESS when the function succeeds

hal_igmp_snooping_add_entry

This API adds a multicast entry for the source, group for a given VLAN. If the group does not exist, a new one is created. If the group exists, the list of ports is added to the entry.

Syntax

```
int hal_igmp_snooping_add_entry (char *bridge_name, struct hal_in4_addr *src, struct
hal_in4_addr *group, char is_exclude, int vid, int svid, int count,u_int32_t
*ifindexes);
```

Input Parameters

`bridge_name`	Bridge Name
`src`	Multicast source address
`group`	Multicast group address
`vlan`	VLAN ID
`count`	Count of ports to add
`ifindexes`	array of ports to add

Output Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_ENTRY_ERR when the multicast entry for the source, group for a given VLAN is not a success

HAL_SUCCESS when the function succeeds

hal_igmp_snooping_delete_entry

This API deletes a multicast entry for a (source, group) for a given VLAN. If the group doesn't exist, an error is returned. If the group exists, the list of ports are deleted from the multicast entry. If it is the last port for the multicast entry, this multicast entry is deleted as well.

Syntax

```
int hal_igmp_snooping_delete_entry (char *bridge_name, struct hal_in4_addr *src, struct
hal_in4_addr *group, char is_exclude, int vid, int svid, int count,u_int32_t
*ifindexes);
```

Input Parameters

`bridge_name`	Bridge Name
`src`	Multicast source address
`group`	Multicast group address
`vlan`	VLAN ID
`count`	Count of ports to add
`ifindexes`	array of ports to add

Output Parameters

None

Return Values

HAL_ERR_IGMP_SNOOPING_ENTRY_ERR when the multicast delete for the source, group for a given VLAN is not a success

HAL_SUCCESS when the function succeeds

hal_mld_snooping_enable

The API enables MLD snooping for the bridge.

Syntax

```
int hal_mld_snooping_enable (char *bridge_name);
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_ENABLE when the MLD snooping for the bridge is not enabled

HAL_SUCCESS when the function succeeds

hal_mld_snooping_disable

The API disables MLD snooping for the bridge.

Syntax

```
int hal_mld_snooping_disable (char *bridge_name);
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_DISABLE when the MLD snooping for the bridge is not disabled

HAL_SUCCESS when the function succeeds

hal_mld_snooping_if_enable

The API enables MLD snooping on an interface.

Syntax

```
hal_mld_snooping_if_enable (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index value

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_ENABLE when the MLD snooping on the interface is not enabled

HAL_SUCCESS when the function succeeds

hal_MLD_snooping_if_disable

The API enables IGMP snooping on an interface.

Syntax

```
hal_mld_snooping_if_disable (char *name, unsigned int ifindex);
```

Input Parameters

`name`	Interface name
`ifindex`	Interface index value

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_DISABLE when the MLD snooping on the bridge is not disabled

HAL_SUCCESS when the function succeeds

hal_mld_snooping_add_entry

This API adds a multicast entry for the source, group for a given VLAN. If the group does not exist, a new one is created. If the group exists, the list of ports is added to the entry.

Syntax

```
int hal_mld_snooping_add_entry (char *bridge_name, struct hal_in4_addr *src, struct
hal_in4_addr *group, char is_exclude, int vid, int svid, int count,u_int32_t
*ifindexes);
```

Input Parameters

`bridge_name`	Bridge Name
`src`	Multicast source address
`group`	Multicast group address
`vlan`	VLAN ID
`count`	Count of ports to add
`ifindexes`	array of ports to add

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_ENTRY_ERR when the multicast entry for the source, group for a given VLAN is not a success

HAL_SUCCESS when the function succeeds

hal_mld_snooping_delete_entry

This API deletes a multicast entry for a (source, group) for a given VLAN. If the group doesn't exist, a error will be returned. If the group exists, the list of ports are deleted from the multicast entry.If it is the last port for the multicast entry, this multicast entry is deleted as well.

Syntax

```
int hal_mld_snooping_delete_entry (char *bridge_name, struct hal_in4_addr *src, struct
hal_in4_addr *group, char is_exclude, int vid, int svid, int count,u_int32_t
*ifindexes);
```

Input Parameters

`bridge_name`	Bridge Name
`src`	Multicast source address
`group`	Multicast group address
`vlan`	VLAN ID
`count`	Count of ports to add
`ifindexes`	array of ports to add

Output Parameters

None

Return Values

HAL_ERR_MLD_SNOOPING_DELETE_ERR when the multicast delete for the source, group for a given VLAN is not a success

HAL_SUCCESS when the function succeeds

Index