ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Network Services Module
Developer Guide
December 2015

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 xi
  Audience xi
  Conventions xi
  Contents xi
  Related Documents xii
  Support xii
  Comments xii

CHAPTER 1 Overview 11
  Introduction 11
    NSM Protocol Messaging 11
    Message Handler 12
    NSM Server 12
    NSM Client 12

CHAPTER 2 Data Structures 13
  Common Data Structures 13
  nsm_client 13
    Definition 14
  Interface Group Data Structures 15
    rcvaddr_index 15
  ARP Data Structure 15
    nsm_if 15
  QoS Data Structures 20
    nsm_msg_qos 20
    nsm_msg_qos_clean 22
    nsm_msg_qos_release 22
    nsm_server_entry 23
    qos_interface 25
  Route Map Data Structures 27
    ipi_vr 27
  Logging Data Structures 29
    zlog 29
  Message Data Structures 31
    nsm_msg_header 31
    nsm_tlv_header 31
    nsm_msg_service 32

CHAPTER 3 Interface Management 35
  Overview 35
    Layer 2 Port 35
    Multiple Layer 2 Ports on a Link 35
    Layer 2 Link Aggregation 36

Layer 2 Interface to Layer 3 Interface 36
Multiple Layer 3 Interfaces on a Layer 2 Interface 36
Layer 3 Interface Aggregation 37
Interface APIs 38
Include File 38
nsm_if_add_update 38
nsm_if_delete_update 39
nsm_if_down 39
nsm_if_loopback_set 39
nsm_if_refresh 40
nsm_if_up 41
nsm_ip_address_install 41
nsm_ip_address_uninstall 42
nsm_ip_address_uninstall_all 43
nsm_ipv6_address_install 43
nsm_ipv6_address_uninstall 44
nsm_ipv6_address_uninstall_all. 45
SNMP linkUp and linkDown Traps 45

CHAPTER 4 Interface Group MIB API 47
Overview 47
Include File 47
nsm_get_rcvaddress_status. 47
nsm_get_rcvaddress_type 48

CHAPTER 5 Internet Protocol Security API 49
Data Structures 49
ipsec_crypto_map_bundle 49
ipsec_crypto_map 50
ipsec_transform_set 51
ipsec_crypto_isakmp 52
Command API 54
Include File 55
ipsec_crypto_isakmp_disable. 55
ipsec_crypto_isakmp_enable 55
ipsec_crypto_map_reset 56
ipsec_esp_session_key_set. 56
ipsec_esp_session_key_unset. 57
ipsec_esp_transform_set_create 57
ipsec_interface_crypto_map_exist 58
ipsec_isakmp_crypto_bundle_reset 58
ipsec_match_address 59
ipsec_peer_address_set 59
ipsec_peer_address_unset 60
ipsec_peer_ipv6_address_set 60
ipsec_peer_ipv6_address_unset 61
ipsec_transform_set_delete 61
ipsec_transformset_set 61

ipsec_transformset_unset 62
ipsec_validate_security_parameters 63

CHAPTER 6 Platform Abstraction Layer 65
Directory Structure 65
PAL API 65
pal_if_mip6_home_agent_set 67
pal_if_mip6_home_agent_unset 67
pal_kernel_fib_create 67
pal_kernel_fib_delete 68
pal_kernel_gratuitous_arp_send 68
pal_kernel_if_bind_vrf 69
pal_kernel_if_flags_get 69
pal_kernel_if_flags_set 69
pal_kernel_if_flags_unset 70
pal_kernel_if_get_bw 70
pal_kernel_if_get_hwaddr 70
pal_kernel_if_get_index 71
pal_kernel_if_get_metric 71
pal_kernel_if_get_mtu 71
pal_kernel_if_info 72
pal_kernel_if_ipv4_address_add 72
pal_kernel_if_ipv4_address_delete 72
pal_kernel_if_ipv4_address_delete_all 73
pal_kernel_if_ipv4_address_secondary_add 73
pal_kernel_if_ipv4_address_secondary_delete 74
pal_kernel_if_ipv4_address_update 74
pal_kernel_if_ipv6_address_add 74
pal_kernel_if_ipv6_address_delete 75
pal_kernel_if_scan 75
pal_kernel_if_unbind_vrf 76
pal_kernel_if_update 76
pal_kernel_ipv4_add 76
pal_kernel_ipv4_del 77
pal_kernel_ipv4_forwarding_get 77
pal_kernel_ipv4_forwarding_set 77
pal_kernel_ipv4_update 78
pal_kernel_ipv6_add 78
pal_kernel_ipv6_del 79
pal_kernel_ipv6_forwarding_get 79
pal_kernel_ipv6_forwarding_set 79
pal_kernel_ipv6_old_del 80
pal_kernel_ipv6_update 80
pal_kernel_L2_ipv4_resolve 81
pal_kernel_L2_ipv6_resolve 81
pal_kernel_route_scan 81
pal_kernel_start 82

pal_kernel_stop 82
pal_kernel_virtual_ipv4_add 83
pal_kernel_virtual_ipv4_delete 83
pal_kernel_virtual_mac_add 83
pal_kernel_virtual_mac_delete 84
pal_kernel_vrrp_start 84

CHAPTER 7 Quality of Service Resource Manager API 87
Overview 87
QoS APIs 87
Include File 88
nsm_qos_serv_clean_for 88
nsm_qos_serv_if_deinit 89
nsm_qos_serv_if_get 89
nsm_qos_serv_if_init 89
nsm_qos_serv_init 90
nsm_qos_serv_update_max_bw 90
nsm_qos_serv_update_max_resv_bw 91
nsm_read_qos_client_clean 91
nsm_read_qos_client_init 92
nsm_read_qos_client_modify 92
nsm_read_qos_client_probe 92
nsm_read_qos_client_release 93
nsm_read_qos_client_reserve 93

CHAPTER 8 MIB-II API 95
Overview 95
SNMP APIs 95
Include File 95
nsm_get_sys_desc 96
nsm_get_sys_location 96
nsm_get_sys_name 96
nsm_get_sys_oid 97
nsm_get_sys_services 97
nsm_get_sys_up_time 98

CHAPTER 9 Address Resolution Protocol API 99
ARP APIs 99
Include Files 99
nsm_api_arp_entry_add 99
nsm_api_arp_entry_del 100
nsm_if_proxy_arp_set 100

CHAPTER 10 Route Map API 103
Overview 103
Route Map APIs 103
Include File 105
route_map_match_as_path_set 106
route_map_match_as_path_unset 106

route_map_match_community_set 107
route_map_match_community_unset 108
route_map_match_interface_set 108
route_map_match_interface_unset 109
route_map_match_ip_address_prefix_list_set 110
route_map_match_ip_address_prefix_list_unset 110
route_map_match_ip_address_set 111
route_map_match_ip_address_unset 112
route_map_match_ip_nexthop_prefix_list_set 112
route_map_match_ip_nexthop_prefix_list_unset 113
route_map_match_ip_nexthop_set 114
route_map_match_ip_nexthop_unset 114
route_map_match_ipv6_address_prefix_list_set. 115
route_map_match_ipv6_address_prefix_list_unset. 116
route_map_match_ipv6_address_set 116
route_map_match_ipv6_address_unset 117
route_map_match_ipv6_nexthop_set 118
route_map_match_ipv6_nexthop_unset 118
route_map_match_metric_set 119
route_map_match_metric_unset 120
route_map_match_origin_set 120
route_map_match_origin_unset 121
route_map_match_route_type_set 122
route_map_match_route_type_unset 122
route_map_match_tag_set 123
route_map_match_tag_unset 124
route_map_set_aggregator_as_set 124
route_map_set_aggregator_as_unset 125
route_map_set_as_path_prepend_set 126
route_map_set_as_path_prepend_unset 126
route_map_set_atomic_aggregate_set 127
route_map_set_atomic_aggregate_unset 128
route_map_set_community_delete_set 128
route_map_set_community_delete_unset 129
route_map_set_community_set. 130
route_map_set_community_unset. 130
route_map_set_dampening_set 131
route_map_set_dampening_unset 132
route_map_set_ext_community_rt_set 132
route_map_set_ext_community_rt_unset 133
route_map_set_ext_community_soo_set 134
route_map_set_ext_community_soo_unset 134
route_map_set_ip_nexthop_set. 135
route_map_set_ip_nexthop_unset. 136
route_map_set_ipv6_nexthop_local_set 136
route_map_set_ipv6_nexthop_local_unset 137
route_map_set_ipv6_nexthop_set. 138

route_map_set_ipv6_nexthop_unset 138
route_map_set_local_preference_set 139
route_map_set_local_preference_unset 140
route_map_set_metric_set 140
route_map_set_metric_type_set 141
route_map_set_metric_type_unset 142
route_map_set_metric_unset 142
route_map_set_originator_id_set 143
route_map_set_originator_id_unset 144
route_map_set_origin_set 144
route_map_set_origin_unset 145
route_map_set_tag_set 146
route_map_set_tag_unset 146
route_map_set_vpnv4_nexthop_set 147
route_map_set_vpnv4_nexthop_unset 148
route_map_set_weight_set 148
route_map_set_weight_unset 149

CHAPTER 11 Logging API. 151
Directory Structure 151
Logging APIs 151
pal_log_close 151
pal_log_open 152
pal_log_output 152
pal_log_start 152
pal_log_stop 153

Appendix A NSM Messages 155
NSM Message Structures 155
NSM Services. 155
NSM Messages. 156
Authentication Messages 157
CFM Messages 158
ELMI Messages 158
G8031 and G8032 Messages 159
Interface Messages 159
ISIS Messages 159
Label Pool Messages 160
Layer-2 Messages 160
LDP Messages 161
Multicast Messages 162
Nexthop Messages. 163
OAM Messages 164
PBB Messages. 165
PVLAN Messages 166
QoS Client Messages. 166
Route Messages 167
VLAN Messages. 167

VR Messages. 168
VRF Messages 168
Miscellaneous Messages. 169
Index 171

Preface

This guide describes the application programming interface (API) for Network Services Module (NSM) in ZebOS-XP.

Audience

This guide is intended for developers who write code to customize and extend NSM.

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 document contains these chapters and appendices:

- Chapter 1, *Overview*
- Chapter 2, *Data Structures*
- Chapter 3, *Interface Management*
- Chapter 4, *Interface Group MIB API*
- Chapter 6, *Platform Abstraction Layer*
- Chapter 7, *Quality of Service Resource Manager API*
- Chapter 8, *MIB-II API*
- Chapter 9, *Address Resolution Protocol API*
- Chapter 10, *Route Map API*
- Chapter 11, *Logging API*
- Appendix A, *NSM Messages*

Related Documents

The following guides are related to this document:

- *Network Services Module Command Reference*
- *Layer 2 Command Reference*
- *Layer 2 Developer Guide*
- *Layer 2 Configuration Guide*
- *Policy Based Routing Configuration 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.

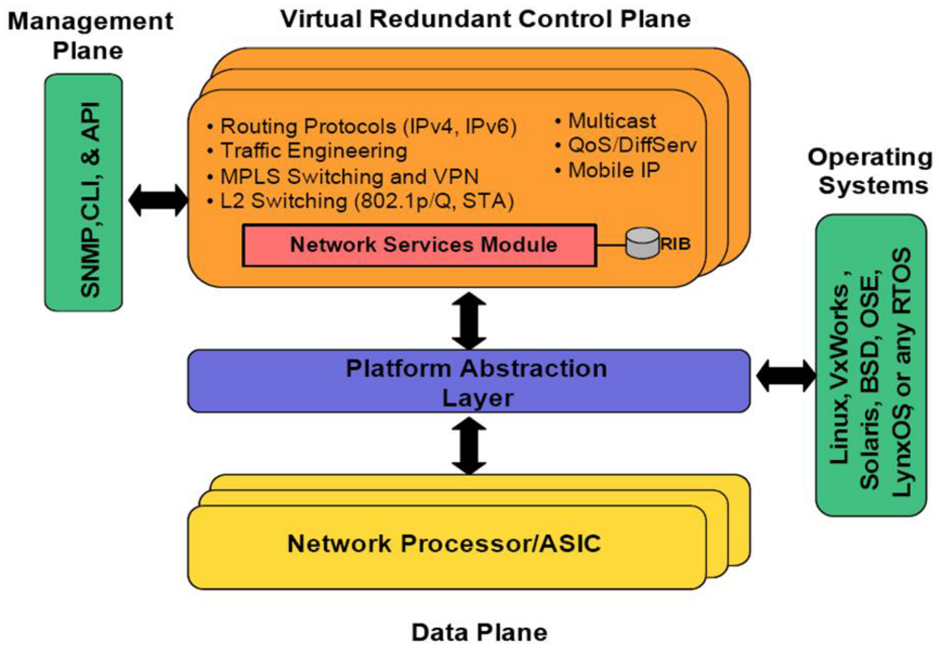
CHAPTER 1 Overview

This chapter introduces NSM and its components.

Introduction

The Network Services Module (NSM) is the base component that simultaneously and independently communicates with every ZebOS-XP routing and switching process. The NSM acts as the backbone of the ZebOS-XP modules and supports IPv4, IPv6, MPLS, Mobile IP, DiffServ extensions, DiffServ-TE, Multicast, and Layer 2/Layer 3 based protocols. In addition, NSM supports routing redundancy, virtual routing, IGMPv1, IGMPv2 and IGMPv3, as well as management and configuration services for all protocol modules.

NSM communicates directly with each ZebOS-XP routing and switching module to manage route tables and to perform route conversion and redistribution. NSM also interfaces with the Platform Abstraction Layer (PAL) to communicate with the underlying operating system, or with the Hardware Abstraction Layer (HAL), to communicate with network processors for forwarding table updates. This powerful architecture gives equipment developers and designers the flexibility and freedom to integrate any of the ZebOS-XP modules with many popular operating systems.



NSM Protocol Messaging

This section describes the messages NSM uses to communicate with other modules. The NSM protocol communication separates the message handling mechanism from NSM protocol handling. The message handling mechanism handles the connection, and the protocol handling is serviced by the respective client. It is composed of three components:

- Message handler
- NSM client

- NSM server

Message Handler

The message handler supports two types of communication mechanisms; UNIX domain sockets and TCP. For each NSM client, one communication channel (such as a socket) is established. For messages such as Nexthop Lookup, QoS Queries, and Label Pool Services, a synchronous response is expected from NSM. Because NSM can send any other response, the client adds these to the pending read queue. This read queue is maintained in the client, and a pending read event is launched after the respective response is received from NSM.

NSM sends most of the asynchronous notifications to protocols. Each of these connection types (synchronous/asynchronous) initiate a connection separately. If any type of connection channel is lost (due to NSM termination), both connections are closed and re-initiated. Certain protocols might not require the synchronous channel.

The message handler registers these callbacks for these actions:

- Connect: Connection is established
- Disconnect: Connection is disconnected
- Read header: Read an NSM message header
- Read message: Read an NSM message

The message handler also maintains the state (connected or disconnected) of the communication channel.

NSM Server

The NSM server maintains the services available for a particular version of ZebOS-XP. It installs parsers and callbacks for the messages received from protocols. The NSM server maintains a vector of clients to which it is connected, maintaining up to a maximum of two NSM server entries for each client. One of them is for the sync and the other for the async connection from the client.

NSM Client

The NSM client is a link to NSM. Each protocol has an instance of the NSM client. The NSM client encompasses the message handler. The protocols initially request the services required from NSM using the NSM client. Depending upon the response from NSM, the protocol decides whether it has sufficient services available to resume or abandon operation. After the NSM client requests the services, NSM responds with all services available for this client.

NSM Client Parser and Callback

For each of the messages received from NSM, the NSM client installs parsers for the messages. Protocols do not have to install the parser functions. The protocols have to register callback functions to be called by the respective parser routine. The typedef of the callback function is as follows:

```
typedef int (*NSM_CALLBACK) (struct nsm_msg_header *, void *, void *);
```

`header`	Pointer to the NSM message header
`arg`	Pointer to the NSM connection handler
`message`	Pointer to the message structure

NSM Client Disconnect Handling

If the protocol NSM client installs the disconnect handler, the client library calls the disconnect handler. When the disconnect handler registers, a reconnection initiates if a loss of connection to NSM occurs. If the disconnection handler is not registered, no reconnections are attempted. In this event, restart both NSM and the protocol.

CHAPTER 2 Data Structures

This chapter describes the data structures and enumerators that are used by NSM.

Common Data Structures

See the *Common Data Structures Developer Guide* for a description of these data structures used by multiple ZebOS-XP modules:

- `connected`
- `interface`
- `lib_globals`
- `nsm_bridge`
- `nsm_bridge_master`
- `nsm_master`
- `pal_in4_addr`
- `pal_in6_addr`
- `prefix`
- `prefix_ipv6`
- `rib`
- `stream`

nsm_client

Any NSM client must declare an instance of this data structure. It is defined in the `lib/nsm_client.h` file.

Member	Description
`zg`	Daemon-specific globals
`service`	Service bits
`client_id`	NSM client ID
`parser`	Parser functions
`callback`	Callback functions
`subcallback`	Sub callback functions for specific PM need
`async`	Asynchronous connection

Member	Description
disconnect_callback	Disconnect callback
t_connect	Reconnect thread
reconnect_interval	Reconnect interval in seconds
debug	Debug message flag
nc_commsg_recv_cb	COMMMSG receive callback
nc_commsg_user_ref	COMMMSG user context

Definition

```
struct nsm_client
{
  struct lib_globals *zg;

  /* Service bits. */
  struct nsm_msg_service service;

  /* NSM client ID. */
  u_int32_t client_id;

  /* Parser functions. */
  NSM_PARSER parser[NSM_MSG_MAX];

  /* Callback functions. */
  NSM_CALLBACK callback[NSM_MSG_MAX];

  /* Async connection. */
  struct nsm_client_handler *async;

  /* NSM shutdown message*/
  u_char nsm_server_flags;
  #define NSM_SERVER_SHUTDOWN  (1<<0)

  /* Disconnect callback. */
  NSM_DISCONNECT_CALLBACK disconnect_callback;

  /* Reconnect thread. */
  struct thread *t_connect;

  /* Reconnect interval in seconds. */
  int reconnect_interval;

  /* Debug message flag. */
  int debug;
  /* COMMMSG recv callback. */
  nsm_msg_commsg_recv_cb_t  nc_commsg_recv_cb;
  void                     *nc_commsg_user_ref;
};
```

Interface Group Data Structures

The following data structure is specific to the interface group function. Refer to the Chapter 4, *Interface Group MIB API* chapter for more information on the interface group APIs.

rcvaddr_index

This data structure is a combination of interface index and MAC address. It is defined in the `nsm/nsm_api.h` file.

Type	Definition
`ifindex`	Interface index
`mac_addr[ETHER_ADDR_LEN]`	MAC address

Definition

```
struct rcvaddr_index
{
  int ifindex;
  u_char mac_addr[ETHER_ADDR_LEN];
};
```

ARP Data Structure

The following data structure is specific to the ARP APIs. Refer to the Chapter 9, *Address Resolution Protocol API* chapter for more information on the ARP APIs.

nsm_if

This data structure holds information about an NSM interface. It is defined in the `nsm/nsm_interface.h` file.

Member	Description
`ifp`	Interface
`type`	Interface type
`acl_name_str`	Access list name pointer
`acl_dir_str`	Access list directory pointer

Member	Description
rtadv_if	Route advertisement interface'
nsm_if_link_changed	Interface status change TLV
vrrp_if	VRRP interface
agg	Aggregator
agg_config_type	The type of aggregator association: STATIC or LACP
agg_mode	Aggregator mode
agg_oper_state	Oper state of the aggregator member
hw_aggregated	Flag to denote that h/w attach for member interface for standby
exp_bridge_grpd	Whether the interface is explicitly bridge grouped or is bridge grouped as a result of belonging to an aggregator
conf_key	If restoration of channel-group command fails, store the key for later restorations
conf_chan_activate	Activate channel; required for possible restoration of channel-group command
conf_agg_config_type	Config type; required for possible restoration of channel-group command
nsm_bridge_port_conf	Store the configuration of aggregator interface
opcode	Opcode for addition, deletion of agg; significant only for members, not for agg
nsm_if_lacp_cdr_ref	Lib connected LACP checkpoint database record reference
nsm_if_lacp_agg_associate_cdr_ref	LACP interface aggregator association CDR reference
vid	Virtual ID
l2_flags	L2 configuration flags
port_vlan	Currently only 32 ports are supported for Port Based VLAN
bridge	Bridge
switchport	Bridge-port
nsm_bw_profile	Pointer which contains the BW Profile Parameters related to this UNI
bridge_static_mac_config	Static MAC configuration to the bridge table
gvrp_port_config	GVRP port configuration
gmrp_port_cfg	GMRP port configuration
mac_acl	MAC access list
group_tree	VLAN classification groups

Member	Description
`qos_if`	QOS interface
`vif`	Virtual interface
`mcast_ttl`	IP multicast time to live threshold value
`mif`	Multicast interface
`flags`	Interface-related flags (for example, is this interface proxy ARP)
`member`	Protocol membership information
`l2mcastif`	Layer 2 multicast interface
`efm_oam_if`	EFM OAM interface
`lldp_oam_if`	LLDP OAM interface
`nsm_if_cdr_ref`	Interface checkpoint database record reference
`nsm_ifma_vec`	Interface MAC address vector
`cctree`	Pointer to control channels which bind to this interface
`nsm_rpf_info`	RPF information

Definition

```
struct nsm_if
{
  struct interface *ifp;
  u_char type;
  char *acl_name_str;
  char *acl_dir_str;
#ifdef HAVE_RTADV
  struct rtadv_if *rtadv_if;
#endif /* HAVE_RTADV */
  u_char nsm_if_link_changed;

#ifdef HAVE_VRRP
  VRRP_IF vrrp_if;
#endif /* HAVE_VRRP */

#ifdef HAVE_LACPD
  struct nsm_if_agg agg;

/* A flag to denote the type of Aggregator Association STATIC or LACP*/
  u_char agg_config_type;
  u_char agg_mode;

  /* flag to store oper state of the aggregator member */
  u_char agg_oper_state;
  /* flag to denote that h/w attach for member interface for standby */
  bool_t hw_aggregated;
/* A flag to denote whether the interface is explicitly bridge grouped
   or it is bridge grouped as a result of belonging to an aggregator
 */
 u_char exp_bridge_grpd;

  /* If restoration of channel-group command fails during boot-up,
     the key should br stored for later trials of restoration
  */
  u_int16_t conf_key;

  /* activate channel; required for possible restoration of
     channel-group command
  */
  bool_t conf_chan_activate;

  /* config type; required for possible restoration of
     channel-group command
  */
  u_char conf_agg_config_type;

/* Store the configuration of aggregator interface */

  struct nsm_bridge_port_conf *nsm_bridge_port_conf;

  /* Opcode for addition, deletion of agg.
   * Significant only for members, not for agg */
  u_char opcode;
#define NSM_LACP_AGG_ADD          1
#define NSM_LACP_AGG_DEL          2

#ifdef HAVE_HA
  HA_CDR_REF nsm_if_lacp_cdr_ref;
  HA_CDR_REF nsm_if_lacp_agg_associate_cdr_ref;
#endif /* HAVE_HA */

#endif /* HAVE_LACP */

  u_int16_t vid;

#ifdef HAVE_L2

#ifdef HAVE_VLAN

#define NSM_VLAN_PORT_BASED_VLAN_ENABLE         (1 << 2)
#define NSM_VLAN_DOT1Q_ENABLE                   (1 << 1)
#define NSM_VLAN_DOT1Q_DISABLE                  (1 << 0)
  u_int16_t l2_flags;
  /* Currently only 32 ports are supported for Port Based VLAN */
  u_int32_t port_vlan;
#endif /* HAVE_VLAN */

  struct nsm_bridge *bridge;

  struct nsm_bridge_port *switchport;

#ifdef HAVE_PROVIDER_BRIDGE
  struct nsm_band_width_profile *nsm_bw_profile; ///< Pointer which contains
                         ///< the BW Profile Parameters related to this UNI
#endif /* HAVE_PROVIDER_BRIDGE */
  struct list *bridge_static_mac_config;

#ifdef HAVE_GVRP
  struct gvrp_port_config *gvrp_port_config;
#endif /* HAVE_GVRP */

#ifdef HAVE_GMRP
  struct gmrp_port_config *gmrp_port_cfg;
#endif /* HAVE_GMRP */

#ifdef HAVE_L2LERN
  struct mac_acl *mac_acl;
#endif /* HAVE_L2LERN */

#ifdef HAVE_VLAN_CLASS
  struct avl_tree *group_tree;
#endif /* HAVE_VLAN_CLASS */
#endif /* HAVE_L2 */

#ifdef HAVE_TE
  struct qos_interface *qos_if;
#endif /* HAVE_TE */

#ifdef HAVE_L3
  u_int32_t flags;
#define NSM_IF_SET_PROXY_ARP            (1 << 0)
#ifdef HAVE_RPF
struct nsm_rpf_info *rpf_info;
#endif /* HAVE_RPF */
#endif
#ifdef HAVE_BFD
#define NSM_IF_BFD                      (1 << 1)
#define NSM_IF_BFD_DISABLE              (1 << 2)
#ifdef HAVE_IPV6
#define NSM_IF_BFD_IPV6                 (1 << 3)
#define NSM_IF_BFD_IPV6_DISABLE         (1 << 4)
#endif /* HAVE_IPV6 */
```

```
#endif
#ifdef HAVE_L3
  /* Flag to see if LDP IGP sync is configured on this interface */
#define LDP_IGP_SYNC_ENABLE_OSPF (1 << 5)
#define LDP_IGP_SYNC_ENABLE_ISIS (1 << 6)
#define LDP_IGP_SYNC_ENABLE \
        (LDP_IGP_SYNC_ENABLE_OSPF | LDP_IGP_SYNC_ENABLE_ISIS)
#endif /* HAVE_L3 */

  /* LDP Session state information */
  u_int32_t ldp_session_state;

  /* Protocol membership information. */
  modbmap_t member;

#if defined HAVE_IGMP_SNOOP || defined HAVE_MLD_SNOOP
  struct nsm_l2_mcast_if l2mcastif;
#endif /* HAVE_IGMP_SNOOP || HAVE_MLD_SNOOP */

#ifdef HAVE_ONMD
  struct nsm_efm_oam_if *efm_oam_if;
  struct nsm_lldp_oam_if *lldp_oam_if;
#endif /* HAVE_ONMD */

#ifdef HAVE_HA
  HA_CDR_REF nsm_if_cdr_ref;
#endif /* HAVE_HA */

  NSM_IFMA_VEC nsm_ifma_vec;

#ifdef HAVE_GMPLS
  /* Pointer to control channels which bind to this interface */
  struct avl_tree cctree;
#endif /* HAVE_GMPLS */
};
```

QoS Data Structures

The following data structures are specific to the QoS resource manager functions. See Chapter 7, *Quality of Service Resource Manager API* for more about the QoS API.

nsm_msg_qos

This data structure holds QoS information. It is locate in the `lib/nsm_message.h` file.

Member	Description
`cindex`	Cindex
`resource_id`	Resource ID
`protocol_id`	Protocol ID
`id`	ID
`owner`	MPLS Owner
`ct_num`	Class Type number for DSTE usage
`setup_priority`	Setup priority
`hold_priority`	Pre-emption specific priority
`t_spec`	Tspec
`if_spec`	Ifspec
`ad_spec`	Adspec
`status`	Status

Definition

```
struct nsm_msg_qos
{
  /* Cindex. */
  cindex_t cindex;

  /* Resource ID. */
  u_int32_t resource_id;

  /* Protocol ID. */
  u_int32_t protocol_id;

  /* ID. */
  u_int32_t id;

  /* MPLS Owner. */
  struct mpls_owner owner;

  /* Class Type number for DSTE usage. */
  u_char ct_num;

  /* Setup priority. */
  u_int8_t setup_priority;

  /* Pre-emption specific priority. */
  u_int8_t hold_priority;
  /* Tspec */
  struct nsm_msg_qos_t_spec t_spec;

  /* Ifspec */
  struct nsm_msg_qos_if_spec if_spec;

  /* Adspec */
  struct nsm_msg_qos_ad_spec ad_spec;

  /* Status */
  u_int32_t status;
};
```

nsm_msg_qos_clean

This data structure is sent by protocols to NSM to clean up QoS. It is defined in the `lib/nsm_message.h` file.

Member	Description
`cindex`	Cindex
`protocol_id`	Protocol ID
`ifindex`	Ifindex

Definition

```
struct nsm_msg_qos_clean
{
  /* Cindex. */
  cindex_t cindex;

  /* Protocol ID. */
  u_int32_t protocol_id;

  /* Ifindex. */
  u_int32_t ifindex;
};
```

nsm_msg_qos_release

This data structure is sent by protocols to NSM to release QOS resources. It is defined in the `lib/nsm_message.h` file.

Member	Function
`cindex`	Cindex
`protocol_id`	Protocol ID
`resource_id`	Resource ID
`ifindex`	Ifindex
`status`	Status

Definition

```
struct nsm_msg_qos_release
{
  /* Cindex. */
  cindex_t cindex;

  /* Protocol ID. */
  u_int32_t protocol_id;

  /* Resource ID. */
  u_int32_t resource_id;

  /* Ifindex. */
  u_int32_t ifindex;

  /* Status. */
  u_int32_t status;
};
```

nsm_server_entry

This data structure holds NSM information stored by each client. It is defined in the `nsm/nsm_server.h` file.

Member	Description
next	Linked list
prev	Linked list
me	Pointer to message entry
ns	Pointer to NSM server structure
nsc	Pointer to NSM server client
service	NSM service structure
send	Send buffer

Member	Description
recv	Receive buffer
buf_ipv4	Message buffer for IPv4 redistribute
pnt_ipv4	IPv4 address pointer
len_ipv4	IPv4 address length
t_ipv4	Ipv4 thread
buf_ipv6	Message buffer for IPv6 redistribute
pnt_ipv6	IPv6 address pointer
len_ipv6	IPv6 address length
t_ipv6	IPv6 thread
send_msg_count	Send message count
recv_msg_count	Received message count
connect_time	Connect time
read_time	Last read time
redist	Redistribute request comes from this client
send_queue	Message queue
t_write	Write thread
message_id	Message ID
last_read_type	For record
last_write_type	Record

Definition

```
struct nsm_server_entry
{
  /* Linked list. */
  struct nsm_server_entry *next;
  struct nsm_server_entry *prev;

  /* Pointer to message entry.  */
  struct message_entry *me;

  /* Pointer to NSM server structure.  */
  struct nsm_server *ns;

  /* Pointer to NSM server client.  */
  struct nsm_server_client *nsc;
  /* NSM service structure.  */
  struct nsm_msg_service service;

  /* Send/Recv buffers. */
  struct nsm_server_entry_buf send;
  struct nsm_server_entry_buf recv;

  /* Message buffer for IPv4 redistribute.  */
  u_char buf_ipv4[NSM_MESSAGE_MAX_LEN];
  u_char *pnt_ipv4;
  u_int16_t len_ipv4;
  struct thread *t_ipv4;

#ifdef HAVE_IPV6
  /* Message buffer for IPv6 redistribute.  */
  u_char buf_ipv6[NSM_MESSAGE_MAX_LEN];
  u_char *pnt_ipv6;
  u_int16_t len_ipv6;
  struct thread *t_ipv6;
#endif /* HAVE_IPV6 */

  /* Send and recieved message count.  */
  u_int32_t send_msg_count;
  u_int32_t recv_msg_count;

  /* Connect time.  */
  pal_time_t connect_time;

  /* Last read time.  */
  pal_time_t read_time;

  /* Redistribute request comes from this client.  */
  struct nsm_redistribute *redist;

  /* Message queue.  */
  struct fifo send_queue;
  struct thread *t_write;

  /* Message id */
  u_int32_t message_id;

  /* For record.  */
  u_int16_t last_read_type;
  u_int16_t last_write_type;
};
```

qos_interface

This data structure old QoS interface information. It is defined in the `nsm/mpls/nsm_qos_serv.h` file.

Member	Description
ifp	Interface
aggr_rsvd_bw	Sum of bandwidths reserved at holding priority q across class types where: q <= p
ct_aggr_rsvd_bw	Sum of reserved bandwidths of all the established LSPs belonging to class CTb with holding priority of q where: TE_Class [i] -- <CTb, p> and q <= p
resource_array	Table of resources keyed on resource id; the array index is based on hold priorities
status	Status of interface

Definition

```
struct qos_interface
{
  struct interface *ifp;
  /*
    aggr_rsvd_bw [p] = sum of bandwidths reserved at
    holding priority q across class types where q <= p
  */
  float32_t aggr_rsvd_bw[MAX_PRIORITIES];

#ifdef HAVE_DSTE
  /*
    sum of reserved bandwidths of all the established LSPs
    belonging to class CTb with holding priority of q where :
    TE_Class [i] -- <CTb, p> and  q <= p
  */
  float32_t ct_aggr_rsvd_bw[MAX_TE_CLASS];
#endif /* HAVE_DSTE */
  /* Table array of resources keyed on resource id */
  /* The array index are based on hold priorities */
  struct route_table *resource_array [MAX_PRIORITIES];

#ifdef HAVE_MPLS_TP
  /* Al preempted resources entities will be added to bw_awaiting_pool indexed as 
below*/
  /* Table array of resources keyed on resource id */
  /* The array index are based on hold priorities */
  struct list *bw_awaiting_pool [MAX_PRIORITIES];
#endif
  /* Status of interface */
#define QOS_INTERFACE_DISABLED     0
#define QOS_INTERFACE_ENABLED      1
  u_char status;
};
```

Route Map Data Structures

The following data structure is specific to the route map functions. Refer to the Chapter 10, *Route Map API* chapter for more information on the route map APIs.

ipi_vr

This data structure holds virtual router information. It is defined in the `lib/lib.h` file.

Member	Description
`zg`	Pointer to globals
`name`	VR name
`id`	VR ID
`router_id`	Router ID
`flags`	VR flag
`ifm`	Interface master
`vrf_vec`	VRFs
`vrf_list`	VRFs
`host`	Host
`access_master_ipv4`	Access list
`access_master_ipv6`	Access list
`prefix_master_ipv4`	Prefix list
`prefix_master_ipv6`	Prefix list
`prefix_master_orf`	Prefix master outbound route filter
`route_match_vec`	Route map match vector
`route_set_vec`	Route map set vector
`route_map_master`	Route map master list
`keychain_list`	Key chain
`proto`	Protocol master
`t_config`	Config read event.
`vrf_in_cxt`	VRF currently in context
`t_if_stat_threshold`	If stats update threshold timer

Member	Description
`entLogical`	Logical entity structure
`mappedPhyEntList`	Entity list
`snmp_community`	Community string to identify current VR
`lib_vr_cdr_ref`	VR checkpoint database record reference library
`pbr_event`	Event for the VR

Definition

```
struct ipi_vr
{
  /* Pointer to globals. */
  struct lib_globals *zg;

  /* VR name. */
  char *name;

  /* VR ID. */
  u_int32_t id;

  /* Router ID. */
  struct pal_in4_addr router_id;

  u_int8_t flags;
#define LIB_FLAG_DELETE_VR_CONFIG_FILE    (1 << 0)

  /* Interface Master. */
  struct if_vr_master ifm;

  /* VRFs. */
  vector vrf_vec;

  /* VRFs. */
  struct ipi_vrf *vrf_list;

  /* Protocol bindings. */
  u_int32_t protos;

  /* Host. */
  struct host *host;

  /* Access List. */
  struct access_master access_master_ipv4;
#ifdef HAVE_IPV6
  struct access_master access_master_ipv6;
#endif /* def HAVE_IPV6 */
 /* Prefix List. */
```

```
  struct prefix_master prefix_master_ipv4;
#ifdef HAVE_IPV6
  struct prefix_master prefix_master_ipv6;
#endif /* HAVE_IPV6 */
  struct prefix_master prefix_master_orf;

  /* Route Map. */
  vector route_match_vec;
  vector route_set_vec;
  struct route_map_list route_map_master;

  /* Key Chain. */
  struct list *keychain_list;

  /* Protocol Master. */
  void *proto;

  /* Config read event. */
  struct thread *t_config;

  /* VRF currently in context */
  struct ipi_vrf *vrf_in_cxt;

  /* If stats update threshold timer */
  struct thread *t_if_stat_threshold;

  struct entLogicalEntry *entLogical;
  struct list *mappedPhyEntList;

  /* Community string to identify current VR */
  struct snmpCommunity snmp_community;

#ifdef HAVE_HA
  HA_CDR_REF lib_vr_cdr_ref;
#endif /* HAVE_HA */

#ifdef HAVE_PBR
  struct pbr_rmap_event pbr_event;
#endif /* HAVE_PBR */
};
```

Logging Data Structures

The following data structure is specific to system logging. Refer to the Chapter 11, *Logging API* chapter for more information on the logging APIs.

zlog

This data structure holds information about a log. It is defined in the `lib/log.h` file.

Member	Description
dest	Log destination
instance	Instance
protocol	Protocol ID
maskpri	Mask priority
record_priority	Priority
flags	Flags
logfile	Log file name
log_maxsize	Maximum size
pal_log_data	Platform specific data

Definition

```
struct zlog
{
  /* Log destination. */
  enum log_destination dest;
  /* Instance. */
  u_int32_t instance;

  /* Protocol ID */
  module_id_t protocol;

  /* Mask priority. */
  u_int32_t maskpri;
  /* Priority. */
  u_int32_t record_priority;

  /* Flags. */
  u_char flags;
#ifdef PAL_LOG_FILESYS
  /* Log filename. */
  char *logfile;
  u_int32_t log_maxsize;
#endif /* PAL_LOG_FILESYS */

  /* Platform specific data. */
  void *pal_log_data;
};
```

Message Data Structures

The following data structures are specific to the NSM Messaging. Refer to the Appendix A, *NSM Messages* chapter for more information on the PAL APIs.

nsm_msg_header

This data structure helps manage NSM context headers. It is defined in the `lib/nsm_message.h` file.

Member	Description
vr_id	Virtual router identifier
vrf_id	VPN routing/forwarding identifier
type	Message type
length	Message length
message_id	Message ID

Definition

```
struct nsm_msg_header
{
  /* VR-ID. */
  u_int32_t vr_id;
  /* VRF-ID. */
  u_int32_t vrf_id;
  /* Message Type. */
  u_int16_t type;
  /* Message Len. */
  u_int16_t length;
  /* Message ID. */
  u_int32_t message_id;
};
```

nsm_tlv_header

This data structure helps manage NSM TLV headers. It is defined in the `lib/nsm_message.h` file.

Member	Description
type	Message type
length	Message length

Definition

```
struct nsm_tlv_header
```

```
{
  u_int16_t type;
  u_int16_t length;
};
```

nsm_msg_service

This data structure helps manage the NSM service message format. It is used by NSM_MSG_SERVICE_REQUEST and NSM_MSG_SERVICE_REPLY. This struct is defined in the `lib/nsm_message.h` file.

Member	Description
cindex	TLV flags
version	NSM Protocol Version.
reserved	Reserved
protocol_id	Protocol ID
client_id	Client ID
bits	Service bits
restart_state	Graceful restart state
restart[AFI_MAX][SAFI_MAX];	Graceful restart TLV.
grace_period	Grace period expires TLV.
restart_val	Restart option TLV
restart_length	Restart length
nsm_msg_label_pool *label_pools	Label pools used before restart
label_pool_num	Label pools number

Definition

```
struct nsm_msg_service
{
  /* TLV flags. */
  cindex_t cindex;

  /* NSM Protocol Version. */
  u_int16_t version;

  /* Reserved. */
  u_int16_t reserved;

  /* Protocol ID. */
  u_int32_t protocol_id;

  /* Client Id. */
  u_int32_t client_id;

  /* Service Bits. */
  u_int32_t bits;

  /* Graceful Restart State */
  u_char restart_state;

  /* Graceful Restart TLV. */
  u_char restart[AFI_MAX][SAFI_MAX];

  /* Grace Perioud Expires TLV. */
  pal_time_t grace_period;

  /* Restart Option TLV.  */
  u_char *restart_val;
  u_int16_t restart_length;

#if (defined HAVE_MPLS || defined HAVE_GMPLS)
  /* Label pools used before restart. */
  struct nsm_msg_label_pool *label_pools;
  u_int16_t label_pool_num;
#endif /* HAVE_MPLS || HAVE_GMPLS */
};
```

CHAPTER 3 Interface Management

This chapter describes how NSM manages hardware interfaces for ZebOS-XP.

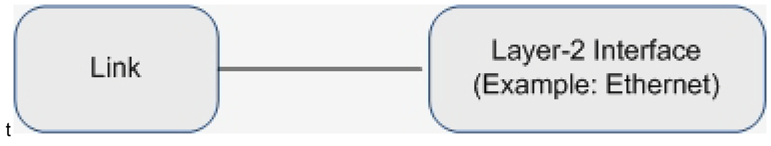
Overview

Routers and switches have one or more physical connections, commonly called links. Links connect one router or switch to other network elements. The data transfer occurs over this physical link. A link has an associated Layer 2 (L2) protocol (for example, PPP, or Ethernet) to transfer packets over the media of the link. It is possible that more than one Layer 2 protocol exists over a link, for example, PPP over Ethernet. Multiple Layer 2 interfaces of the same type can be bundled together to form a Layer 2 (802.3ad) link aggregation. A Layer 2 interface is typically referred to as a port. A Layer 2 interface maintains the Layer 2 properties of the link, for example, maximum transmission unit (MTU), speed, duplex, in case of Ethernet links.

Multiple Layer-3 interfaces can be created on top of Layer 2 ports. The Layer-3 interface maintains the Layer-3 properties, for example, IP addresses, prefix length, MTU. Multiple hierarchies are possible r the different interfaces supported typically on a Layer 2 switch, Layer-3 router, or a Hybrid Layer 2/Layer-3 Switch-Router (software router). The images that follow illustrate these capabilities.

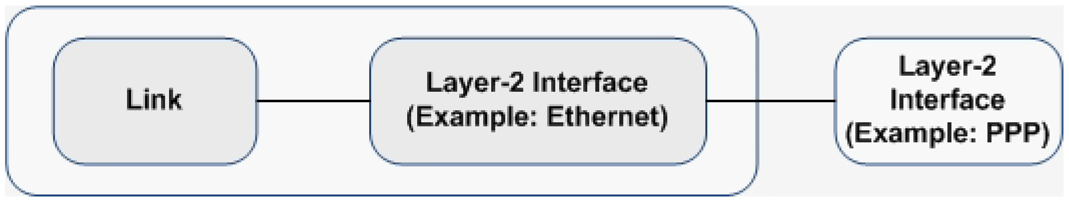
Layer 2 Port

The figure below illustrates a single Layer 2 Ethernet interface.



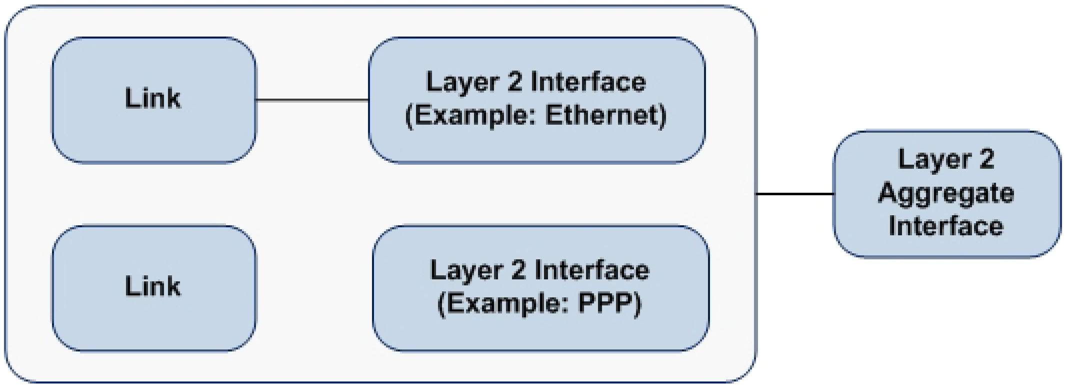
Multiple Layer 2 Ports on a Link

The figure below illustrates a link with two interfaces; one could be an Ethernet interface and the second one could be a PPP interface.


Link
Layer-2 Interface (Example: Ethernet)
Layer-2 Interface (Example: PPP)

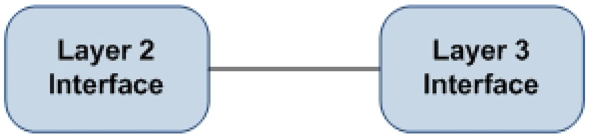
Layer 2 Link Aggregation

The figure below illustrates two links and two Layer 2 interfaces aggregated to a single Layer 2 interface.


Link
Layer 2 Interface (Example: Ethernet)
Link
Layer 2 Interface (Example: PPP)
Layer 2 Aggregate Interface

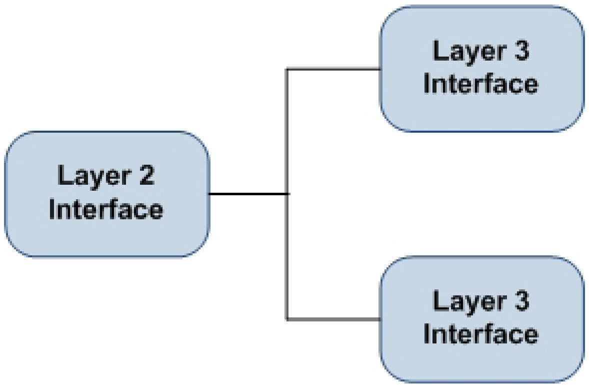
Layer 2 Interface to Layer 3 Interface

The figure below illustrates a Layer 2 interface linking with a Layer 3 interface.



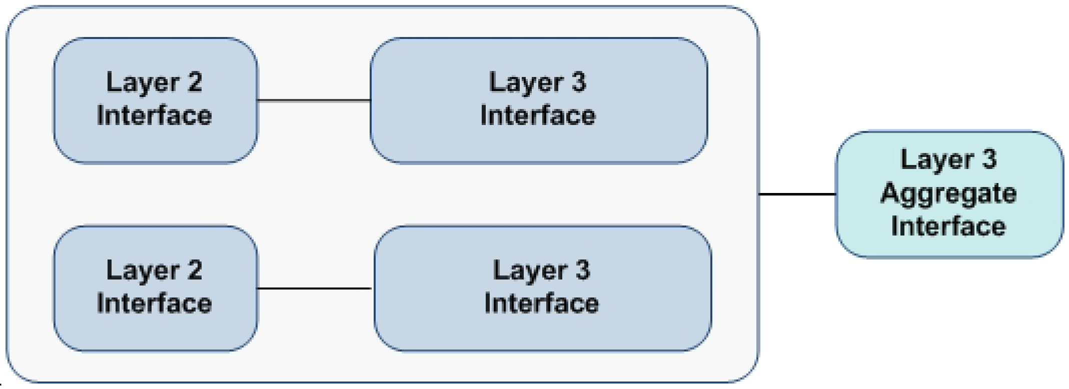
Multiple Layer 3 Interfaces on a Layer 2 Interface

The figure below illustrates multiple Layer 3 interfaces connecting to a single Layer 2 interface.

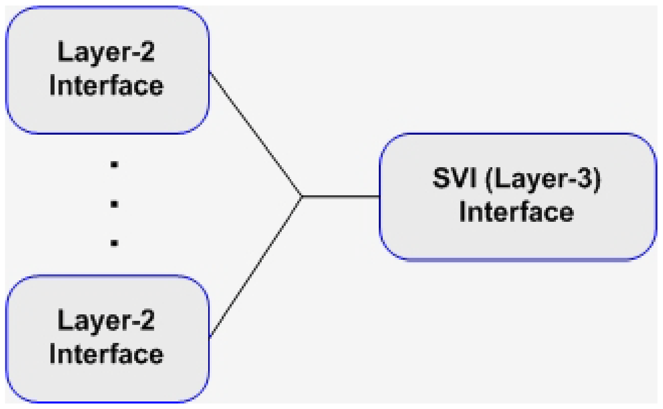


Layer 3 Interface Aggregation

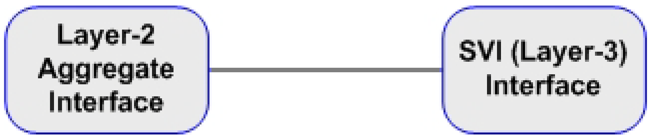
The figure below illustrates two Layer 2 interfaces and two Layer 3 interfaces aggregated to single Layer 3 interface.



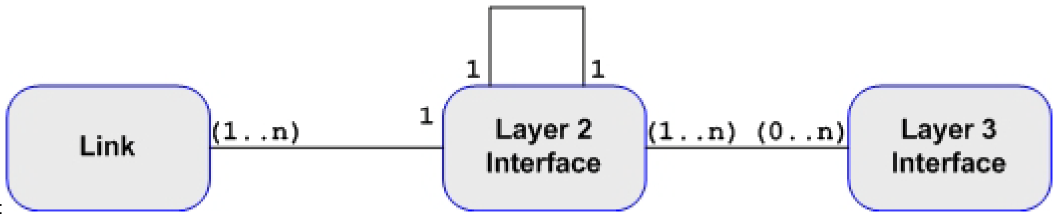
Switch VLAN Interface (IP Interface)



Switch VLAN Interface on Aggregated Layer 2 Port



Overall Link Relationships



Interface management in the Hardware Services Layer (HSL) provides a set of APIs, and implementation for the configuration and management of physical and logical interfaces.

Interface APIs

This section documents the functions used to manage interfaces.

Function	Description
nsm_if_add_update	Adds an interface
nsm_if_delete_update	Deletes an interface
nsm_if_down	Takes down an interface
nsm_if_loopback_set	Enables or disables loopback for a given interface
nsm_if_refresh	Refreshes an interface
nsm_if_up	Brings up an interface
nsm_ip_address_install	Installs an IP address
nsm_ip_address_uninstall	Uninstalls an IP address
nsm_ip_address_uninstall_all	Uninstalls all IP dresses on a given interface
nsm_ipv6_address_install	Installs an IPv6 address
nsm_ipv6_address_uninstall	Uninstalls an IPv6 address
nsm_ipv6_address_uninstall_all	Uninstalls all IPv6 address on a given interface

Include File

To call the functions in this chapter, you must include `nsm/nsm_interface.h`.

nsm_if_add_update

This function adds an interface.

Syntax

```
void
nsm_if_add_update (struct interface *ifp, fib_id_t fib_id)
```

Input Parameters

`ifp`	Pointer to interface
`fib_id`	Forwarding table ID

Output Parameters

None

Return Values

None

nsm_if_delete_update

This function deletes an interface.

Syntax

```
void
nsm_if_delete_update (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to interface

Output Parameters

None

Return Values

None

nsm_if_down

This function takes down an interface.

Syntax

```
void
nsm_if_down (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to interface

Output Parameters

None

Return Values

None

nsm_if_loopback_set

This function enables or disables loopback for a given interface.

This function implements the `loopback (disable|enable)` command.

Syntax

```
int
nsm_if_loopback_set(struct interface *ifp,
                    bool_t enable_flag)
```

Input Parameters

`ifp`	Pointer to interface
`enable_flag`	Whether to enable or disable loopback; one of these constants from `nsm/nsmd.h`:
`NSM_TRUE`	Enable loopback
`NSM_FALSE`	Disable loopback

Output Parameters

None

Return Values

NSM_ERR_INVALID_INTERFACE when the interface is NULL

NSM_ERR_IF_NOT_FOUND when the NSM interface cannot be found

NSM_ERR_IF_LOOPBACK_SET when trying to enable loopback for an interface where it is already enabled

NSM_ERR_IF_LOOPBACK_UNSET when trying to disable loopback for an interface where it is already disabled

NSM_ERR_IF_VLAN when the interface is for a VLAN

NSM_ERR_IF_RUNNING when the interface is running

NSM_ERR_IF_PMIRROR_SET when the interface is configured for mirroring

NSM_ERR_IF_AGGREGATED when the interface is aggregated

NSM_ERR_IF_EGRESS_SET when the interface is an egress port

NSM_ERR_IF_REDIRECT_SET when the interface is configured for redirect

NSM_ERR_IF_STATIC_ARP_SET when ARP loopback is set for the interface

NSM_ERR_MEM_ALLOC_FAILURE when memory allocation fails

NSM_FAILURE when the function fails

NSM_SUCCESS when the function succeeds

nsm_if_refresh

This function refreshes an interface.

Syntax

```
void
nsm_if_refresh (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to interface

Output Parameters

None

Return Values

None

nsm_if_up

This function brings up an interface.

Syntax

```
void
nsm_if_up (struct interface *ifp)
```

Input Parameters

`ifp`	Pointer to interface

Output Parameters

None

Return Values

None

nsm_ip_address_install

This function installs an IP address. In addition, it implements the `ip address A.B.C.D/M (secondary|)` command.

Syntax

```
int
nsm_ip_address_install (u_int32_t vr_id, char *ifname,
                        struct pal_in4_addr *addr, u_char prefixlen,
                        char *peer_str, int secondary, int vrrp)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`addr`	IPv4 address
`prefixlen`	Prefix length; one of these constants from `lib/prefix.h`:

`IN_CLASSA_PREFIXLEN`

`IN_CLASSB_PREFIXLEN`

`IN_CLASSC_PREFIXLEN`

`peer_str`	This parameter is ignored
`secondary`	Whether the address is secondary
`vrrp`	Whether the address was set by VRRP

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_INVALID_IPV4_ADDRESS when the given IP address is experimental or is a class D address

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist or the prefix length is invalid

NSM_API_SET_ERR_CANT_SET_ADDRESS_ON_P2P when the interface is point to point

NSM_API_SET_ERR_CANT_SET_ADDRESS_WITH_ZERO_IFINDEX when the interface index is zero (0)

NSM_API_SET_ERR_CANT_SET_SECONDARY_FIRST when adding a secondary address to the interface before adding the primary address

NSM_API_SET_ERR_MAX_ADDRESS_LIMIT when the interface has the maximum number of addresses

NSM_API_SET_ERR_CANT_CHANGE_PRIMARY when the given address is primary, but the `secondary` parameter is true

NSM_API_SET_ERR_CANT_CHANGE_SECONDARY when the given address is secondary, but the `secondary` parameter is false

NSM_API_SET_ERR_SAME_ADDRESS_EXIST if the given address already exists

NSM_API_SET_ERR_ADDRESS_OVERLAPPED when the given address overlaps

NSM_API_SET_ERR_CANT_SET_ADDRESS when there is a problem adding the address to the hardware

CLI_SUCCESS when the function succeeds

nsm_ip_address_uninstall

This function uninstalls an IP address.

This function implements the `no ip address A.B.C.D/M (secondary|)` command.

Syntax

```
int
nsm_ip_address_uninstall (u_int32_t vr_id, char *ifname,
                          struct pal_in4_addr *addr, u_char prefixlen,
                          char *peer_str, int secondary, int vrrp)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`addr`	IPv4 address
`prefixlen`	Prefix length; one of these constants from `lib/prefix.h`:

```
IN_CLASSA_PREFIXLEN
IN_CLASSB_PREFIXLEN
IN_CLASSC_PREFIXLEN
```

`peer_str`	This parameter is ignored
`secondary`	Whether the address is secondary
`vrrp`	Whether the address was set by VRRP

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist or the prefix length is invalid

NSM_API_SET_ERR_ADDRESS_NOT_EXIST when the IPv4 interface is unnumbered, the address does not exist, or the address was set by VRRP

NSM_API_SET_ERR_CANT_CHANGE_PRIMARY when the given address is primary, but the `secondary` parameter is true

NSM_API_SET_ERR_MUST_DELETE_SECONDARY_FIRST when the given address is primary, but a secondary address still exists

NSM_API_SET_SUCCESS when the function succeeds

nsm_ip_address_uninstall_all

This function uninstalls all IP addresses on a given interface.

This function implements the `no ip address` command.

Syntax

```
int
nsm_ip_address_uninstall_all (u_int32_t vr_id, char *ifname)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist

NSM_API_SET_SUCCESS when the function succeeds

nsm_ipv6_address_install

This function installs an IPv6 address.

This function implements the `ipv6 address X:X::X:X/M` command.

Syntax

```
int
nsm_ipv6_address_install (u_int32_t vr_id, char *ifname,
                          struct pal_in6_addr *addr, u_char prefixlen,
                          char *peer_str, char *label, int anycast, int vrrp)
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface
`addr`	IPv6 address
`prefixlen`	Prefix length; one of these constants from `lib/prefix.h`:

`IN_CLASSA_PREFIXLEN`

`IN_CLASSB_PREFIXLEN`

`IN_CLASSC_PREFIXLEN`

`peer_str`	This parameter is ignored
`label`	Label of the address
`anycast`	Whether this is an anycast address
`vrrp`	Whether the address was set by VRRP

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist or the prefix length is invalid

NSM_API_SET_ERR_IF_ADDR_MAX_PER_IFP when the maximum number of addresses per interface is exceeded

NSM_API_SET_ERR_IF_ADDR_MAX when the maximum number of addresses is exceeded

NSM_API_SET_ERR_INVALID_IPV4_ADDRESS_VRRP when a VRRP session is configured with this address

NSM_API_SET_ERR_CANT_SET_ADDRESS_VRRP when the interface is set for VRRP, but the `vrrp` parameter is false

NSM_API_SET_ERR_ADDRESS_OVERLAPPED when the given address overlaps

NSM_API_SET_SUCCESS when the function succeeds

nsm_ipv6_address_uninstall

This function uninstalls an IPv6 address.

This function implements the `no ipv6 address X:X::X:X/M` command.

Syntax

```
int
nsm_ipv6_address_uninstall (u_int32_t vr_id, char *ifname,
                            struct pal_in6_addr *addr, u_char prefixlen, int vrrp)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`addr`	IPv6 address
`prefixlen`	Prefix length; one of these constants from `lib/prefix.h`:

`IN_CLASSA_PREFIXLEN`

`IN_CLASSB_PREFIXLEN`

`IN_CLASSC_PREFIXLEN`	
`vrrp`	Whether the address was set by VRRP

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist or the prefix length is invalid

NSM_API_SET_ERR_ADDRESS_NOT_EXIST when the IPv6 interface is unnumbered, the address does not exist, or the address was set by VRRP

NSM_API_SET_ERR_CANT_UNSET_ADDRESS_VRRP when the address was set by VRRP

NSM_API_SET_SUCCESS when the function succeeds

nsm_ipv6_address_uninstall_all

This function uninstalls all IPv6 address on a given interface.

Syntax

```
int
nsm_ipv6_address_uninstall_all (u_int32_t vr_id, char *ifname)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master does not exist

NSM_API_SET_ERR_IF_NOT_EXIST if the interface does not exist

NSM_API_SET_SUCCESS when the function succeeds

SNMP linkUp and linkDown Traps

RFC 2863 defines linkUp and linkDown traps:

- linkUp means that a communication link has left the down state and transitioned to some other state
- linkDown means that a communication link is about to enter the down state from some other state

NSM sends a linkUp or linkDown trap when a network interface comes up or goes down respectively.

CHAPTER 4 Interface Group MIB API

This chapter describes the support for RFC 2863 (Interface Group MIB).

Overview

RFC 2863 defines these variables that are supported by NSM:

Object Type	Syntax	Access	Functions
ifRcvAddressStatus	RowStatus	read-create	nsm_get_rcvaddress_status
ifRcvAddressType	INTEGER	read-create	nsm_get_rcvaddress_type

Include File

To call the functions in this chapter, you must include `nsm/nsm_api.h`.

nsm_get_rcvaddress_status

This function gets the rcvaddress status.

Syntax

```
int
nsm_get_rcvaddress_status (struct rcvaddr_index *rcvaddr, int *status)
```

Input Parameters

`rcvaddr`	Pointer to `rcvaddr_index` structure

Output Parameters

`status`	Rcvaddr status

Return Values

NSM_API_GET_SUCCESS

RESULT_ERROR

NSM_BRIDGE_ERR_NOT_BOUND

NSM_API_GET_ERROR

nsm_get_rcvaddress_type

This function checks whether the mac address is static or dynamic.

Syntax

```
int
nsm_get_rcvaddress_type (struct rcvaddr_index *rcvaddr, int *type)
```

Input Parameters

`rcvaddr`	Pointer to `rcvaddr_index` structure

Output Parameters

`type`	Rcvaddress type

Return Values

Always NSM_API_GET_SUCCESS

CHAPTER 5 Internet Protocol Security API

This chapter describes the support for Internet Protocol Security (IP security).

Data Structures

The data structures in this section are defined in `pal/linux/pal_ipsec.h`.

ipsec_crypto_map_bundle

This data structure maintains Crypto-map SA bundle.

Member	Description
`name`	Name of the Crypto bundle.
`mode`	The mode of the bundle.
`flags`	Crypto bundle flags.
`local_addr_ifname`	Name of the interface to be used as local address for SA traffic.

Definition

```
struct ipsec_crypto_map_bundle
{
  /* Crypto map bundle name
   * All crypto maps will be considered as bundles. The list crypto map
   * will house more than one crypto maps structure and when this
   * crypto-map bundle is applied to the interface and if it has the list
   * populated  we will group these maps else we will process it as usual.
   */
  char name[IPSEC_CONFIG_NAME_LEN];
  struct list *crypto_map;
  /* The interface on which this bundle is aplied.*/
  /* Specify the interface on which it is applied */
  struct list *if_list;
  /* The mode of the bundle */
  u_char mode;
#define IPSEC_ISAKMP        (1 << 0)
#define IPSEC_MANUAL        (1 << 1)
  /* Flag to denote if no has been issued on this bundle or not */
  u_int32_t flags;
#define IPSEC_NO_CRYPTO_MAP_FLAG                (1 << 0)
#define IPSEC_CRYPTO_BUNDLE_LOCAL_ADDR          (1 << 1)
```

```
  /* Name of the interface to used as local address for SA traffic*/
  char local_addr_ifname[INTERFACE_NAMSIZ + 1];
};
```

ipsec_crypto_map

This data structure maintains Cryto Map Structure to hold Crypto Map Information.

Member	Description
`seq_num`	Unique sequence number that identifies a crypto-map.
`bundle_name`	Name of the bundle of which, crypto-map is a part.
`sec_lifetime`	Security-association lifetime.
`accesslist_id`	Access list identifier.
`ipv6_acl_name`	IPv6 Access list name.
`spi`	Security parameter Index.
`spi_out`	Holds the SPI of outbound session key.
`flags`	Crypto map related flags.

Definition

```
struct ipsec_crypto_map
{

  /* Unique sequence no that identifies a crypto-map and sets its priority*/
  int seq_num;
  /* Name of the bundle of which this crypto-map is a part */
  char bundle_name[IPSEC_CONFIG_NAME_LEN];


  /* Specify which transform set should be used
     (Only one transform set can be specified when IKE is not used.)
  */
  /* List of transform-sets so that they can be used on priority basis */
  struct list *transform_set;

  /*  security-association lifetime */
  int sec_lifetime;
  int byte_lifetime; /* Not supported Currently */

  /* Id of Access list that     holds ipsec traffic details */
  u_int32_t accesslist_id;

#ifdef HAVE_IPV6
```

```
  char *ipv6_acl_name;
#endif /* HAVE_IPV6 */

  /* Peer address */
  struct list *peer_addr;
  /* Ah session key list */
  struct list *ah_session_key;
  /* ESP session key list */
  struct list *esp_session_key;

  /* Security parameter Index  */
  u_int32_t spi;

  /* For static SA use only */
  u_int32_t spi_out;

  u_int32_t flags;
#define IPSEC_NO_CRYPTO_MAP_FLAG                    (1 << 0)
#define IPSEC_NO_CRYPTO_MAP_PEER_FLAG               (1 << 1)
#define IPSEC_NO_CRYPTO_MAP_MATCH_FLAG              (1 << 2)
#define IPSEC_NO_CRYPTO_MAP_TRANSFORMSET_FLAG       (1 << 3)
#define IPSEC_NO_CRYPTO_MAP_SESSION_KEY_FLAG        (1 << 4)
#define IPSEC_NO_CRYPTO_MAP_IPV6_MATCH_FLAG         (1 << 5)
};
```

ipsec_transform_set

This data structure holds Transform Set Information.

Member	**Description**
name	Tranform Set Name.
mode	The mode of negotiation.
protocol	IPsec Protocol Flag.
ah_transform	AH Authentication Algorithm Flag.
esp_enc_transform	ESP Encryption Algorithm Flag.
esp_auth_transform	ESP Authentication Algorithm Flag.
ref_cnt	Refrence count.
crypto_unset_flag	Flag to show the status of the crypto map.

Definition

```
struct ipsec_transform_set
{
  /* Tranform Set Name */
  char name[IPSEC_CONFIG_NAME_LEN];

  /* The mode of negotiation */
  /* Currently only support IPSEC_TUNNEL_MODE */
  u_char mode;
  #define IPSEC_TUNNEL_MODE     (1 << 0)
  #define IPSEC_TRANSPORT_MODE  (1 << 1)

  /* IPsec Protocol Flag */
  int protocol;
  #define PROTO_NONE   0
  #define PROTO_UNSPEC 1
  #define PROTO_AH     2
  #define PROTO_ESP    3      /*Default*/

  /*AH Authentication Algorithm Flag */
  int ah_transform;
  #define TRANSFORM_AUTH_NONE        0
  #define TRANSFORM_AUTH_MD5         1
  #define TRANSFORM_AUTH_SHA1        2

  /*ESP Encryption Algorithm Flag */
  int esp_enc_transform;
  #define TRANSFORM_ESP_ENC_NONE          0     /* SADB_EALG_NONE */
  #define TRANSFORM_ESP_ENC_3DES          3     /* SADB_EALG_3DESCBC */
  #define TRANSFORM_ESP_ENC_CAST          6     /* SADB_X_EALG_CASTCBC */
  #define TRANSFORM_ESP_ENC_BLOWFISH      7     /* SADB_X_EALG_BLOWFISHCBC */
  #define TRANSFORM_ESP_ENC_BLOWFISH192  8     /* not match in pfkeyv2.h */
  #define TRANSFORM_ESP_ENC_BLOWFISH256  9     /* not match in pfkeyv2.h */
  #define TRANSFORM_ESP_ENC_AES           12    /* SADB_X_EALG_AESCBC Default */
  #define TRANSFORM_ESP_ENC_AES192        31    /* not match in pfkeyv2.h */
  #define TRANSFORM_ESP_ENC_AES256        32    /* not match in pfkeyv2.h */

  /*ESP Authentication Algorithm Flag */
  int esp_auth_transform;
  #define TRANSFORM_ESP_AUTH_NONE         0
  #define TRANSFORM_ESP_AUTH_MD5          2
  #define TRANSFORM_ESP_AUTH_SHA1         3     /*Default*/

  /* Refrence count */
  u_int16_t ref_cnt;

  int crypto_unset_flag;
};
```

ipsec_crypto_isakmp

This data structure maintains IKE parameters.

Member	Description
`policy_priority`	Policy Priority.
`version`	IPsec IKE version.
`key`	Pre-shared key.
`authentication`	Type or mode of authentication.
`encrypt_algo`	Encryption algorithm.
`lifetime`	IPsec IKE lifetime value.
`group`	Group algorithm..
`hash_algo`	Hash algorithm.
`af`	Address family.
`peer_pubkey`	Peer Public key string.
`local_key_label`	Name of the key label.
`peer_name`	Peer name

Definition

```
struct ipsec_crypto_isakmp
{
  int policy_priority;

#define IPSEC_IKE_VERSION_1               1
#define IPSEC_IKE_VERSION_2               2
#define IPSEC_IKE_VERSION_1_2             3
#define IPSEC_IKE_VERSION_DEFAULT         IPSEC_IKE_VERSION_1_2
  u_int8_t version;

  char key[128];
  int authentication;
  int encrypt_algo;
#define IPSEC_IKE_LIFETIME_DEFAULT        10800
  int lifetime;
  int group;
  int hash_algo;

  struct prefix listen_addr;

  /* Address family */
  u_int8_t af;

  union {
    struct prefix_ipv4 addr;
#ifdef HAVE_IPV6
    struct prefix_ipv6 ipv6_addr;
#endif /*HAVE_IPV6*/
    } peer_addr;
  char *peer_name;
#define IPSEC_ISAKMP_ENABLED                (1 << 0)
  int flag;
  /* Key label */
  char *local_key_label;

#define IPSEC_PEER_PUBKEY_BUF_LEN            1024
  /* Peer Pubkey string */
  char *peer_pubkey;
};
```

Command API

This section contains the IP security functions for ZebOS-XP:

Functions	Description
ipsec_crypto_isakmp_disable	Disables IKE at the peer.
ipsec_crypto_isakmp_enable	Enables IKE at the peer.
ipsec_crypto_map_reset	Resets the cryptographic map.
ipsec_esp_session_key_set	Sets the Security parameter Index (SPI) Encryption Key and Authentication key for Encapsulating security header (ESP) Security Association (SA).
ipsec_esp_session_key_unset	Unset the Security parameter Index (SPI) Encryption Key and Authentication key for Encapsulating security header (ESP) Security Association (SA).
ipsec_esp_transform_set_create	Creates a Transform Set with the specified ESP Encryption and Authentication algorithm.
ipsec_interface_crypto_map_exist	Deletes the cryptographic map entry or set.
ipsec_isakmp_crypto_bundle_reset	Clears active IKE connections.
ipsec_match_address	Sets the access list ID to Cryptographic Map.
ipsec_peer_address_set	Sets the peer address to the Cryptographic Map.
ipsec_peer_address_unset	Unset the peer address.
ipsec_peer_ipv6_address_set	Sets the IPv6 peer address to the Cryptographic Map.
ipsec_peer_ipv6_address_unset	Unset the IPv6 peer address.

Functions	Description
ipsec_transform_set_delete	Deletes a transform set.
ipsec_transformset_set	Defines a transform set.
ipsec_transformset_unset	Unset the transform set form the crypto-map link.
ipsec_validate_security_parameters	Validates the security parameters.

Include File

To call the functions in this chapter, you must include `nsm/ipsec/ipsec_api.h`.

ipsec_crypto_isakmp_disable

This function disable IKE at the local system.

Syntax

```
int
ipsec_crypto_isakmp_disable (u_int32_t vr_id,
                              struct ipsec_crypto_isakmp *isakmp);
```

Input Parameters

`isakmp`	ISAKMP value
`vr_id`	Virtual router ID.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_crypto_isakmp_enable

This function enable IKE at the local system.

Syntax

```
s_int32_t
ipsec_crypto_isakmp_enable (u_int32_t vr_id, struct ipsec_crypto_isakmp *isakmp)
```

Input Parameters

`vr_id`	Virtual router ID.
`isakmp`	ISAKMP value

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_crypto_map_reset

This function reset the cryptographic bundle.

Syntax

```
s_int32_t
ipsec_crypto_map_reset (u_int32_t vr_id,
                        struct ipsec_crypto_map_bundle *crypto_bundle)
```

Input Parameters

`vr_id`	Virtual ID.
`crypto_bundle`	Cryptographic bundle list.

Output Parameters

None

Return Values

IPSEC_ERR_VR_NOT_FOUND when virtual route does not exist.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_esp_session_key_set

This function sets the Security parameter Index (SPI) Encryption Key and Authentication key for Encapsulating security header (ESP) Security Association (SA).

Syntax

```
s_int32_t
ipsec_esp_session_key_set (struct ipsec_crypto_map *crypto_map, s_int32_t spi,
                           char esp_enc_key[128], char esp_auth_key[32],
                           u_char flow)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`spi`	Security parameter Index value
`esp_enc_key[128]`	ESP encryption key length.
`esp_auth_key[32]`	ESP authentication key length.

`flow`	Traffic flow.

Output Parameters

None

Return Values

IPSEC_ERR_MEM_ALLOCATION when memory allocation fails.

IPSEC_ERR_SESSION_KEY_SET_EXIST when session key already set.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_esp_session_key_unset

This function unset the Security parameter Index (SPI) Encryption Key and Authentication key for Encapsulating security header (ESP) Security Association (SA).

Syntax

```
s_int32_t
ipsec_esp_session_key_unset (struct ipsec_crypto_map *crypto_map, u_char flow)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`flow`	Traffic flow.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_esp_transform_set_create

This function creates a Transform Set with the specified ESP Encryption and Authentication algorithm.

Syntax

```
struct ipsec_transform_set *
ipsec_esp_transform_set_create (u_int32_t vr_id, char
transform_set_name[IPSEC_CONFIG_NAME_LEN],
                                s_int32_t esp_auth_type,
                                s_int32_t esp_enc_type)
```

Input Parameters

`vr_id`	Virtual router ID.
`IPSEC_CONFIG_NAME_LEN`	Transform set length (128).

`esp_auth_type`	ESP authentication type.
`esp_enc_type`	ESP encryption type.

Output Parameters

None

Return Values

ipsec_transform_set structure when the function is successful.

NULL when the function fails.

ipsec_interface_crypto_map_exist

This function checks the cryptographic map entry on the interface.

Syntax

```
s_int32_t
ipsec_interface_crypto_map_exist (u_int32_t vr_id,
                                  struct ipsec_crypto_map_bundle *cbundle,
                                  struct interface *ifp)
```

Input Parameters

`vr_id`	Virtual ID.
`cbundle`	Crypto map bundle name
`ifp`	Interface name.

Output Parameters

None

Return Values

TRUE when the function is successful.

FALSE when the function fails.

ipsec_isakmp_crypto_bundle_reset

This function resets active IKE connections.

Syntax

```
void
ipsec_isakmp_crypto_bundle_reset (struct ipsec_crypto_map_bundle *crypto_bundle,
s_int32_t action)
```

Input Parameters

`crypto_bundle`	Crypto map bundle list.
`action`	IP security action.

Output Parameters

None

Return Values

None

ipsec_match_address

This function sets the access list ID to Cryptographic Map.

Syntax

```
s_int32_t
ipsec_match_address (struct cli *cli, struct ipsec_crypto_map *crypto_map,
                        char *access_list_id, afi_t afi,
                         u_char type)
```

Input Parameters

`cli`	CLI tree.
`crypto_map`	Cryptographic map, used to setup IPsec SA.
`access_list_id`	Access list ID.
`afi`	Address family.
`type`	Security Association Database (SADB) type.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_peer_address_set

This function sets the peer address to the Cryptographic Map.

Syntax

```
s_int32_t
ipsec_peer_address_set (struct ipsec_crypto_map *crypto_map,
                        struct pal_in4_addr addr)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`addr`	IPv4 address.

Output Parameters

None

Return Values

IPSEC_ERR_MEM_ALLOCATION when memory allocation fails.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_peer_address_unset

This function unset the peer address.

Syntax

```
s_int32_t
ipsec_peer_address_unset (struct ipsec_crypto_map *crypto_map,
                          struct pal_in4_addr addr)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`addr`	IPv4 address.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_peer_ipv6_address_set

This function sets the IPv6 peer address to the Cryptographic Map.

Syntax

```
s_int32_t
ipsec_peer_ipv6_address_set (struct ipsec_crypto_map *crypto_map,
                             struct pal_in6_addr addr)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`addr`	IPv6 address.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

IPSEC_ERR_MEM_ALLOCATION when memory allocation fails.

ipsec_peer_ipv6_address_unset

This function unset the IPv6 peer address

Syntax

```
s_int32_t
ipsec_peer_ipv6_address_unset (struct ipsec_crypto_map *crypto_map,
                               struct pal_in6_addr addr)
```

Input Parameters

`crypto_map`	Cryptographic map, used to setup IPsec SA.
`addr`	IPv6 address.

Output Parameters

None

Return Values

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_transform_set_delete

This function deletes a transform set.

Syntax

```
s_int32_t
ipsec_transform_set_delete (u_int32_t vr_id, char *transform_set_name)
```

Input Parameters

`vr_id`	Virtual router ID.
`transform_set_name`	Transform set name.

Output Parameters

None

Return Values

IPSEC_ERR_TRANSFORM_SET_NOT_FOUND when transform set with this name does not exist.

IPSEC_ERR_TRANSFORMSET_INUSE when transform set is in use.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_transformset_set

This function defines a transform set.

Syntax

```
s_int32_t
ipsec_transformset_set (u_int32_t vr_id, struct ipsec_crypto_map *crypto_map,
                        char *transform_set_name)
```

Input Parameters

`vr_id`	Virtual router ID.
`crypto_map`	Cryptographic map, used to setup IPsec SA.
`transform_set_name`	Transform set name.

Output Parameters

None

Return Values

IPSEC_ERR_VR_NOT_FOUND when virtual router not found.

IPSEC_ERR_IPSEC_MASTER_NOT_FOUND when IP security master not found.

IPSEC_ERR_TRANSFORM_SET_NOT_FOUND when transform set not found.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_transformset_unset

This function unset the transform set form the crypto-map link.

Syntax

```
s_int32_t
ipsec_transformset_unset (u_int32_t vr_id, struct ipsec_crypto_map *crypto_map,
                          char *transform_set_name)
```

Input Parameters

`vr_id`	Virtual router ID.
`crypto_map`	Cryptographic map, used to setup IPsec SA.
`transform_set_name`	Transform set name.

Output Parameters

None

Return Values

IPSEC_ERR_VR_NOT_FOUND when virtual router not found.

IPSEC_ERR_TRANSFORM_SET_NOT_FOUND when transform set not found.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

ipsec_validate_security_parameters

This function validates the security parameters.

Syntax

```
s_int32_t
ipsec_validate_security_parameters (struct ipsec_crypto_map_bundle
                                           *crypto_bundle,
                                     struct interface *ifp)
```

Input Parameters

`crypto_bundle`	Crypto map bundle list.
`ifp`	Interface name.

Output Parameters

None

Return Values

IPSEC_ERR_CRYPTO_MAP_BUNDLE_NOT_FOUND when crypto map bundle not found.

IPSEC_ERR_TRANSFORM_SET_NOT_LINKED when transform set is not linked to the Crypto Map.

IPSEC_ERR_SPI_NOT_SET when security parameter index not set.

IPSEC_ERR_PEER_NOT_SET when peer not set.

IPSEC_ERR_AUTH_KEY_NOT_SET when Authentication Key not set.

IPSEC_ERR_ENC_KEY_NOT_SET when encryption key not set.

IPSEC_ERR_AUTH_ENC_NOT_SET Authentication encryption not set.

IPSEC_API_SUCCESS when the function is successful.

IPSEC_API_ERROR when there is an error.

CHAPTER 6 Platform Abstraction Layer

This chapter describes the Plat from Abstraction Layer (PAL) functions that update the Forwarding Information Base (FIB) in the operating system kernel.

Directory Structure

Each subdirectory in the `pal` directory is an implementation of PAL for a specific operating system platform. The files in the `api` subdirectory are for baseline reference; the files in other directories provide implementation for them.

PAL API

This section describes each function in the FIB PAL API.

Function	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 interface's maximum transmission unit
pal_kernel_if_info	Sets an IPv4 address, mask, and broadcast address for an interface
pal_kernel_if_ipv4_address_add	Removes an IPv4 address, mask, and broadcast address from an interface

Function	Description
pal_kernel_if_ipv4_address_delete	Removes all IPv4 addresses from an interface
pal_kernel_if_ipv4_address_delete	Removes all IPv4 addresses from an interface
pal_kernel_if_ipv4_address_delete_all	Removes all IPv4 addresses from an interface
pal_kernel_if_ipv4_address_secondary_add	Adds 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
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_L2_ipv6_resolve	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

Function	Description
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_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	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

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	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_fib_create

This function creates a FIB in the forwarding plane.

API Call

```
result_t pal_kernel_fib_create (fib_id_t fib_id);
```

Input Parameters

`fib_id`	FIB ID

Output Parameters

None

Result Value

Platform dependent

pal_kernel_fib_delete

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

API Call

```
result_t pal_kernel_fib_delete (fib_id_t fib_id);
```

Input Parameters

`fib_id`	FIB ID

Output Parameters

None

Result Value

Platform dependent

pal_kernel_gratuitous_arp_send

This function sends the given 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`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

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 *, fib_id_t fib_id);
```

Input Parameters

`ifp`	Pointer to the interface
`fib_id`	VR context ID

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_flags_get

This function gets the flags for an interface, and write the current value to the flags in the interface structure. PAL implementation must also specify the bit positions for the flags.

API Call

```
result_t pal_kernel_if_flags_get (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_flags_set

This function sets an interface flag and updates 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`	Pointer to the interface
`flag`	Flag to set

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`flag`	Flag to unset

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_get_bw

This function gets the bandwidth and writes it to the `interface` structure.

API Call

```
result_t pal_kernel_if_get_bw (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_get_index

This function gets the index for a interface.

API Call

```
result_t pal_kernel_if_get_index (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_get_metric

This function gets the interface metric.

API Call

```
result_t pal_kernel_if_get_metric (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_get_mtu

This function gets the interface maximum transmission unit (MTU).

API Call

```
result_t pal_kernel_if_get_mtu (struct interface *ifp);
```

Input Parameter

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_info

This function gets the information about an interface.

API Call

```
result_t pal_kernel_if_info (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_ipv4_address_add

This function gets 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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the top of connected addresses

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc_old`	Pointer to the connected address to delete
`ifc_new`	Pointer to the connected address to add

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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`	Pointer to the interface
`ifc`	Pointer to the connected address

Output Parameters

None

Result Value

Platform dependent

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

Result Value

Platform dependent

pal_kernel_if_unbind_vrf

This function unbinds an interface from a VR in the dataplane.

API Call

```
result_t
pal_kernel_if_unbind_vrf (struct interface *ifp, fib_id_t table)
```

Input Parameters

`ifp`	Pointer to the interface
`table`	FIB table

Output Parameters

None

Result Value

Platform dependent

pal_kernel_if_update

This function scans the kernel interface list and updates interfaces.

API Call

```
void pal_kernel_if_update (void);
```

Input Parameters

None

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv4_add

This function adds a new entry to the kernel IPv4 forwarding table.

API Call

```
result_t pal_kernel_ipv4_add (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	Pointer to the prefix
`r`	Pointer to the RIB entry

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv4_del

This function removes an existing entry from the kernel IPv4 forwarding table.

API call

```
result_t pal_kernel_ipv4_del (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	Pointer to the prefix
`r`	Pointer to the RIB entry

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv4_forwarding_get

This function checks the current 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`	Pointer to the state

Output Parameters

None

Result Value

Platform dependent

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`	State; non-zero means on

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv4_update

This function updates an existing 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`	Pointer to the prefix
`r`	Pointer to the current RIB entry
`s`	Pointer to the new RIB entry

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv6_add

This function adds a new entry to the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_add (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	Pointer to the prefix
`r`	Pointer to the RIB entry

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv6_del

This function removes an existing entry from the kernel IPv6 forwarding table.

API Call

```
result_t pal_kernel_ipv6_del (struct prefix *p, struct rib *r);
```

Input Parameters

`p`	Pointer to the prefix
`r`	Pointer to the RIB entry

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv6_forwarding_get

This function checks the current 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`	Pointer to the state

Result Value

Platform dependent

pal_kernel_ipv6_forwarding_set

This function turns on IPv6 forwarding in the kernel.

API Call

```
result_t pal_kernel_ipv6_forwarding_set (s_int32_t state);
```

Input Parameters

`state`	State; non-zero means on

Output Parameters

None

Result Value

Platform dependent

pal_kernel_ipv6_old_del

This function removes an existing 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

Result Value

Platform dependent

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`	Pointer to the prefix to update
`r`	Pointer to the current RIB entry
`s`	Pointer to the new RIB entry

Output Parameters

None

Result Value

Platform dependent

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`	Instance
`ip_addr`	IPv4 address to resolve

Output Parameters

`l2_addr`	Pointer to the layer 2 address

Output Parameters

None

Result Value

Platform dependent

pal_kernel_L2_ipv6_resolve

This function resolves an IPv6 address into a layer 2 address.

API Call

```
result_t pal_kernel_L2_ipv6_resolve (u_int32_t instance);
```

Input Parameters

`instance`	Instance
`ip_addr`	IPv6 address to resolve.

Output Parameters

`l2_addr`	Pointer to the layer 2 address

Output Parameters

None

Result Value

Platform dependent

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

Result Value

Platform dependent

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

Result Value

Platform dependent

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

Result Value

Platform dependent

pal_kernel_virtual_ipv4_add

This function adds a virtual IP address has been added 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`	Pointer to the interface
`owner`	Owner status of this address
`vrid`	VRRP virutal router ID

Output Parameters

None

Result Value

Platform dependent

pal_kernel_virtual_ipv4_delete

This function deletes a virtual IP address from the specified 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`	Pointer to the interface
`owner`	Owner status of this address
`vrid`	VRRP Virtual Router ID

Output Parameters

None

Result Value

Platform dependent

pal_kernel_virtual_mac_add

This function adds a virtual MAC address to the given interface. This MAC address is specified in RFC 3678 to be `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`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

pal_kernel_virtual_mac_delete

This function deletes a virtual MAC address from the given interface. This MAC address is specified in RFC 3678 to be `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`	Pointer to the interface

Output Parameters

None

Result Value

Platform dependent

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

Result Value

Platform dependent

CHAPTER 7 Quality of Service Resource Manager API

This chapter describes the Quality of Service Resource Manager (QRM) API in ZebOS-XP.

Overview

The ZebOS-XP Quality of Service Resource Manager (QRM) provides admission control services to MPLS. The services provided by the QRM include the following:

Global bandwidth management

QRM provides a consolidated view of the bandwidth available on a system for MPLS. This reduces the overhead of interface-specific and protocol-specific bandwidth management.

Bandwidth probe support

QRM supports resource querying services by clients to determine the reservable bandwidth available for a specific interface at a specified preemption priority. This allows protocols to make policy based decisions prior to signaling downstream LSPs.

LSP resource reservation/modification support

In addition to resource reservation capabilities, QRM supports modification of pre-existing resource reservations on the fly. This feature reduces IPCs between the QRM and protocols.

Resource preemption support

QRM keeps track of reserved resources, and available bandwidth for reservation for a specified interface at a specified preemption priority. The QRM has the ability to determine that a new reservation request may result in preemption of pre-existing resource reservations. In this instance, the QRM removes the affected reservations, and sends a preemption message to the relevant protocols.

Multiple preemption priority support

QRM supports eight distinct preemption priorities for resource reservations, ranging from 0 to 7. The lower numbers indicate higher priorities. These priorities are used for admission control across resources.

QoS APIs

This section contains the QoS functions for ZebOS-XP.

Function	Description
nsm_qos_serv_clean_for	Cleans up one or all protocols
nsm_qos_serv_if_deinit	Deinitializes an interface
nsm_qos_serv_if_get	Gets the QoS interface

Function	Description
nsm_qos_serv_if_init	Initializes an interface
nsm_qos_serv_init	Initializes the QOS module
nsm_qos_serv_update_max_bw	Sets the maximum bandwidth
nsm_qos_serv_update_max_resv_bw	Sets the maximum reservable bandwidth
nsm_read_qos_client_clean	Cleans out all resources reserved by QRM
nsm_read_qos_client_init	Receives an initialization from a client
nsm_read_qos_client_modify	Modifies a resource
nsm_read_qos_client_probe	Checks whether the required bandwidth is available
nsm_read_qos_client_release	Receives a release from a resource
nsm_read_qos_client_reserve	Reserves a resource for a client

Include File

To call the functions in this chapter, you must include the file `nsm/mpls/nsm_qos_serv.h`.

nsm_qos_serv_clean_for

This function cleans up one or all protocols. If `ifindex` is zero, clean up all affected interfaces; otherwise clean the indicated interface only.

Syntax

```
void
nsm_qos_serv_clean_for (struct nsm_master *nm,
                        u_char protocol, u_int32_t ifindex)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`protocol`	Protocol ID
`ifindex`	Interface index

Output Parameters

None

Return Value

None

nsm_qos_serv_if_deinit

This function deinitializes an interface. The interface is identified by name or index.

Syntax

```
void
nsm_qos_serv_if_deinit (struct interface *ifp, bool_t _delete,
                        bool_t remove_mpls_rib,
                        bool_t send_update)
```

Input Parameters

`ifp`	Pointer to interface structure
`_delete`	Delete if TRUE
`send_update`	Whether to notify clients

Output Parameters

None

Return Value

None

nsm_qos_serv_if_get

This function gets the QoS interface.

Syntax

```
struct qos_interface *
nsm_qos_serv_if_get (struct nsm_master *nm, u_int32_t ifindex);
```

Input Parameters

`nm`	Pointer to the NSM master structure
`ifindex`	Interface index

Output Parameters

None

Return Value

A pointer to the interface when the function succeeds

A NULL pointer when the interface is not found

nsm_qos_serv_if_init

This function initializes an interface.

Syntax

```
struct qos_interface *
nsm_qos_serv_if_init (struct interface *ifp);
```

Input Parameters

`ifp`	Pointer to the interface structure

Output Parameters

None

Return Value

A pointer to the interface when the function succeeds

A NULL pointer when the interface is not found

nsm_qos_serv_init

This function initializes the QOS module.

Syntax

```
void
nsm_qos_serv_init (struct nsm_master *nm);
```

Input Parameters

`nm`	Pointer to the NSM master structure

Output Parameters

None

Return Value

None

nsm_qos_serv_update_max_bw

This function sets the maximum bandwidth. If an instance of this function decreases the bandwidth for an interface, this function preempts all LSPs that have a rate higher than the new maximum bandwidth.

Syntax

```
int
nsm_qos_serv_update_max_bw (struct interface *ifp, float32_t bandwidth);
```

Input Parameters

`ifp`	Pointer to interface data
`bandwidth`	The bandwidth value

Output Parameters

None

Return Value

Zero when the function succeeds

Negative one (-1) when the function fails

nsm_qos_serv_update_max_resv_bw

This function sets the maximum reservable bandwidth. If an instance of this function decreases the bandwidth for an interface, this function preempts all LSPs that have a rate higher than the new maximum bandwidth.

Syntax

```
int
nsm_qos_serv_update_max_resv_bw (struct interface *ifp, float32_t bandwidth,
                                 bool_t is_explicit);
```

Input Parameters

`ifp`	Pointer to interface data
`bandwidth`	The bandwidth
`is_explicit`	If false, the default for the maximum reservable bandwidth is set by `nsm_qos_serv_update_max_bw`

Output Parameters

None

Return Value

Zero (0) when the function succeeds

Negative value when the function fails

nsm_read_qos_client_clean

This function cleans out all resources reserved by QRM.

Syntax

```
int
nsm_read_qos_client_clean (struct nsm_msg_header *header, void *arg,
                           void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct
`message`	A pointer to `nsm_msg_qos_clean` struct

Output Parameters

None

Return Value

Always zero (0)

nsm_read_qos_client_init

This function receives an initialization from a client.

Syntax

```
int
nsm_read_qos_client_init (struct nsm_msg_header *header, void *arg, void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct
`message`	Protocol ID

Output Parameters

None

Return Value

Always zero (0)

nsm_read_qos_client_modify

This function modifies a resource.

Syntax

```
int
nsm_read_qos_client_modify (struct nsm_msg_header *header, void *arg, void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct
`message`	A pointer to `nsm_msg_qos` struct

Output Parameters

None

Return Value

Zero (0) when the function succeeds

Negative value when the function fails

nsm_read_qos_client_probe

This function checks whether the required bandwidth is available.

Syntax

```
int
nsm_read_qos_client_probe (struct nsm_msg_header *header, void *arg,
                              void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct
`message`	A pointer to `nsm_msg_qos` struct

Output Parameters

None

Return Value

Zero (0) when the function succeeds

Negative value when the function fails

nsm_read_qos_client_release

This function receives a release from a resource. This function tries to find the resource by first checking the type contained in the message parameter. If the type is not valid, the function ends, returning 0. With a valid type, the function searches all tables for the resource.

Syntax

```
int
nsm_read_qos_client_release (struct nsm_msg_header *header, void *arg, void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct
`message`	A pointer to `nsm_msg_qos_release` struct

Output Parameters

None

Return Value

Always zero (0)

nsm_read_qos_client_reserve

This function reserves a resource for a client.

Syntax

```
int
nsm_read_qos_client_reserve (struct nsm_msg_header *header, void *arg, void *message);
```

Input Parameters

`header`	A pointer to the NSM message header
`arg`	A pointer to `nsm_server_entry` struct

`message`	A pointer to `nsm_msg_qos` struct

Output Parameters

None

Return Value

Zero (0) when the function succeeds

Negative value when the function fails

CHAPTER 8 MIB-II API

This chapter describes the support for RFC 1213 (Management Information Base for Network Management of TCP/IP-based internets: MIB-II) in NSM.

Overview

NSM supports these variables in the system group defined n RFC 1213:

Object Type	Syntax	Access	Functions
sysDescr	DisplayString	read-only	nsm_get_sys_desc
sysLocation	DisplayString	read-write	nsm_get_sys_location
sysName	DisplayString	read-write	nsm_get_sys_name
sysObjectID	OBJECT IDENTIFIER	read-only	nsm_get_sys_oid
sysServices	INTEGER	read-only	nsm_get_sys_services
sysUpTime	TimeTicks	read-only	nsm_get_sys_up_time

SNMP APIs

This section contains the functions in SNMP APIs.

Function	Description
nsm_get_sys_desc	Gets the full name and version identification of the system
nsm_get_sys_location	Gets the physical location of the node
nsm_get_sys_name	gets the fully-qualified domain name of the nod
nsm_get_sys_oid	Gets the object identifier
nsm_get_sys_services	Gets the set of services that the system offers
nsm_get_sys_up_time	Returns the time since the network management portion of the system was initialized

Include File

To call the functions in this chapter, you must include `nsm/nsm_api.h`.

nsm_get_sys_desc

This function gets the full name and version identification of the system.

Syntax

```
int
nsm_get_sys_desc (char *sysDesc)
```

Input Parameters

None

Output Parameters

`sysDesc`	System description

Return values

NSM_SUCCESS when the function succeeds

NSM_ERR_UNAME_FAILED when the function fails

nsm_get_sys_location

This function gets the physical location of the node.

Syntax

```
int
nsm_get_sys_location(char *location)
```

Input Parameters

None

Output Parameters

`location`	System location

Return Values

Always NSM_SUCCESS

nsm_get_sys_name

This function gets the fully-qualified domain name of the node.

Syntax

```
int
nsm_get_sys_name (char *sysname)
```

Input Parameters

None

Output Parameters

`sysname`	System name

Return Values

Always NSM_SUCCESS

nsm_get_sys_oid

This function gets the object identifier.

Syntax

```
int
nsm_get_sys_oid (oid *id);
```

Input Parameters

None

Output Parameters

`id`	System object identifier

Return values

Always NSM_SUCCESS

nsm_get_sys_services

This function gets the set of services that a system offers: data link, gateway, and/or IP host.

Syntax

```
int
nsm_get_sys_services (int *service)
```

Input Parameters

None

Output Parameters

`service`	The bits that are 1 in this integer indicate the services, with the least significant bit always being zero: Bit 1 - Dalalink Bit 2 - Gateway Bit 3 - IP host

Return Values

Always NSM_SUCCESS

nsm_get_sys_up_time

This function returns the time (in hundredths of a second) since the network management portion of the system was initialized.

Syntax

```
pal_time_t
nsm_get_sys_up_time ()
```

Input Parameters

None

Output Parameters

None

Return Values

The difference between the current time and the start time when the function succeeds

Zero (0) when the function fails

CHAPTER 9 Address Resolution Protocol API

This chapter describes the Address Resolution Protocol (ARP) API for ZebOS-XP which is based on RFC 826 (An Ethernet Address Resolution Protocol).

ARP APIs

This section contains the ARP functions.

Function	Description
nsm_api_arp_entry_add	Creates a static proxy ARP entry
nsm_api_arp_entry_del	Deletes an ARP entry
nsm_if_proxy_arp_set	Enables or disables proxy ARP support

Include Files

To call the functions in this chapter, you must include one or both of these files:

- `nsm/nsm_api.h`
- `nsm/nsm_fea.h`

nsm_api_arp_entry_add

This function creates a static proxy ARP entry. This function implements the `arp A.B.C.D MAC` command.

Syntax

```
#include "nsm/nsm_api.h"
int
nsm_api_arp_entry_add (struct nsm_master *nm, struct pal_in4_addr *addr,
                       unsigned char *mac_addr, struct connected *ifc,
                       struct interface *ifp, struct nsm_if *lpbk_ifp,
                       u_int8_t is_proxy_arp)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`addr`	IP address
`mac_addr`	Hardware address
`ifc`	Directly connected route
`ifp`	Pointer to the interface
`lpbk_ifp`	Pointer to the NSM interface

`is_proxy_arp`	Whether this is a proxy ARP entry

Output Parameters

None

Return Values

NSM_SUCCESS when the function succeeds

NSM_API_SET_ERR_INVALID_VALUE

NSM_API_SET_ARP_GENERAL_ERR

NSM_API_SET_ERR_MAX_STATIC_ARP_LIMIT_EXCEEDED

NSM_API_SET_ERR_MAX_VR_STATIC_ARP_LIMIT_EXCEEDED

nsm_api_arp_entry_del

This function deletes an ARP entry. This function implements the `no arp A.B.C.D MAC` command.

Syntax

```
#include "nsm/nsm_api.h"
int
nsm_api_arp_entry_del (struct nsm_master *nm, struct pal_in4_addr *addr,
                      unsigned char *mac_addr, struct interface *ifp)
```

Input Parameters

`nm`	Pointer to the NSM master structure
`addr`	IP address
`mac_addr`	Hardware address
`ifp`	Pointer to the interface

Output Parameters

None

Return Values

NSM_API_SET_ERR_INVALID_VALUE

NSM_API_SET_ARP_GENERAL_ERR

nsm_if_proxy_arp_set

This function enables or disables proxy ARP support for the interface and implements the `ip proxy-arp` and `no ip proxy-arp` commands.

Syntax

```
#include "nsm/nsm_fea.h"
int
nsm_if_proxy_arp_set (struct interface *ifp, int proxy_arp)
```

Input Parameters

`ifp`	Pointer to the interface
`proxy_arp`	Whether to enable or disable proxy ARP

Output Parameters

None

Return Values

RESULT_NO_SUPPORT

CHAPTER 10 Route Map API

This chapter describes the route-map API for ZebOS-XP.

Overview

ZebOS-XP maintains an identical route-map configuration for these modules:

- IMI
- NSM
- RIP
- RIPng
- OSPFv2
- OSPFv3
- BGP

All route-map functions are defined in one place and each protocol module calls these common function implementations. This chapter describes these route-map functions.

Route Map APIs

This section contains the route map functions in ZebOS-XP.

Function	Description
route_map_match_as_path_set	Sets the match BGP AS path list
route_map_match_as_path_unset	Unsets the match BGP AS path list
route_map_match_community_set	Sets the match BGP community list
route_map_match_community_unset	Unsets the match BGP community list
route_map_match_interface_set	Sets the match first hop interface of a route
route_map_match_interface_unset	Unsets the match first hop interface of a route
route_map_match_ip_address_prefix_list_set	Sets the match address of a route
route_map_match_ip_address_prefix_list_unset	Unsets the match address of a route
route_map_match_ip_address_set	Sets the match address of a route
route_map_match_ip_address_unset	Unsets the match address of a route
route_map_match_ip_nexthop_prefix_list_set	Sets the match next-hop address of a route

Function	Description
route_map_match_ip_nexthop_prefix_list_unset	Unsets the match next-hop address of a route
route_map_match_ip_nexthop_set	Sets the match next-hop address of a route
route_map_match_ip_nexthop_unset	Unsets the match next-hop address of a route
route_map_match_ipv6_address_prefix_list_set	Sets the match address of a route
route_map_match_ipv6_address_prefix_list_unset	Unsets the match address of a route
route_map_match_ipv6_address_set	Sets the match IPv6 address of a route
route_map_match_ipv6_address_unset	Unsets the match IPv6 address of a route
route_map_match_ipv6_nexthop_set	Sets the match IPv6 next-hop address of a route
route_map_match_ipv6_nexthop_unset	Unsets the match IPv6 next-hop address of a route
route_map_match_metric_set	Sets the match metric of a route
route_map_match_metric_unset	Unsets the match metric of a route
route_map_match_origin_set	Sets the BGP origin code
route_map_match_origin_unset	Unsets the BGP origin code
route_map_match_route_type_set	Sets the external route type
route_map_match_route_type_unset	Unsets the external route type
route_map_match_tag_set	Sets a tag value
route_map_match_tag_unset	Unsets a tag value
route_map_set_aggregator_as_set	Sets the AS number of an aggregator
route_map_set_aggregator_as_unset	Unsets the AS number of an aggregator
route_map_set_as_path_prepend_set	Sets the prepend string for a BGP AS-path attribute
route_map_set_as_path_prepend_unset	Unsets the prepend string for a BGP AS-path attribute
route_map_set_atomic_aggregate_set	Sets the BGP atomic aggregate attribute
route_map_set_atomic_aggregate_unset	Unsets the BGP atomic aggregate attribute
route_map_set_community_delete_set	Sets the BGP community list for deletion
route_map_set_community_delete_unset	Unsets the BGP community list for deletion
route_map_set_community_set	Sets the BGP community attribute
route_map_set_community_unset	Unsets the BGP community attribute
route_map_set_dampening_set	Sets route-flap dampening
route_map_set_dampening_unset	Unsets route-flap dampening
route_map_set_ext_community_rt_set	Sets the route target extended community

Function	Description
route_map_set_ext_community_rt_unset	Unsets the route target extended community
route_map_set_ext_community_soo_set	Sets the site-of-origin extended community
route_map_set_ext_community_soo_unset	Unsets the site-of-origin extended community
route_map_set_ip_nexthop_set	Sets the IP address of the next hop
route_map_set_ip_nexthop_unset	Unsets the IP address of the next hop
route_map_set_ipv6_nexthop_local_set	Sets the IPv6 next-hop address
route_map_set_ipv6_nexthop_local_unset	Unsets the IPv6 next-hop address
route_map_set_ipv6_nexthop_set	Sets the IPv6 next-hop address
route_map_set_ipv6_nexthop_unset	Unsets the IPv6 next-hop address
route_map_set_local_preference_set	Sets the BGP local preference path attribute
route_map_set_local_preference_unset	Unsets the BGP local preference path attribute
route_map_set_metric_set	Sets the metric route-map
route_map_set_metric_type_set	Sets the metric route-map
route_map_set_metric_type_unset	Unsets the metric route-map
route_map_set_metric_unset	Unsets the metric value for a destination routing protocol
route_map_set_originator_id_set	Sets the BGP originator ID attribute
route_map_set_originator_id_unset	Unsets the BGP originator ID attribute
route_map_set_origin_set	Sets the BGP origin code
route_map_set_origin_unset	Unsets the BGP origin code
route_map_set_tag_set	Sets the tag value for a destination routing protocol
route_map_set_tag_unset	Unsets the tag value for a destination routing protocol
route_map_set_vpnv4_nexthop_set	Sets the VPNv4 next-hop address
route_map_set_vpnv4_nexthop_unset	Unsets the VPNv4 next-hop address
route_map_set_weight_set	Sets the BGP weight for a routing table
route_map_set_weight_unset	Unsets the BGP weight for a routing table

Include File

To call the functions in this chapter, you must include `lib/api.h`.

route_map_match_as_path_set

This function sets the match BGP AS path list. This function implements the `match as-path` command.

Syntax

```
int
route_map_match_as_path_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds.

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_as_path_unset

This function unsets the match BGP AS path list. This function implements the `no match as-path` command.

Syntax

```
int
route_map_match_as_path_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route

`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_community_set

This function sets the match BGP community list. This function implements the `match community set` command.

Syntax

```
int
route_map_match_community_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_community_unset

This function unsets the match BGP community list. This function implements the `no match community` command.

Syntax

```
int
route_map_match_community_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_interface_set

This function sets the match first hop interface of a route. This function implements the `match interface` command.

Syntax

```
int
route_map_match_interface_set (struct ipi_vr *vr, char *name,
                               int permit, int pref, char *arg)
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route

`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_interface_unset

This function unsets the match first hop interface of a route. This function implements the `no match interface` command.

Syntax

```
int
route_map_match_interface_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_address_prefix_list_set

This function sets the match address of a route. This function implements `match ip address prefix-list` command.

Syntax

```
int
route_map_match_ip_address_prefix_list_set (struct ipi_vr *vr, char *name, int permit,
int preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_address_prefix_list_unset

This function unsets the match address of a route. This function implements the `no match ip address prefix-list` command.

Syntax

```
int
route_map_match_ip_address_prefix_list_unset (struct ipi_vr *vr, char *name, int
permit, int preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:

`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_address_set

This function sets the match address of a route. This function implements the `match ip address` command.

Syntax

```
int
route_map_match_ip_address_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_address_unset

This function unsets the match address of a route. This function implements the `no match ip address` command.

Syntax

```
int
route_map_match_ip_address_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_nexthop_prefix_list_set

This function sets the match next-hop address of a route. This function implements the `match ip nexthop prefix list` command.

Syntax

```
int
route_map_match_ip_nexthop_prefix_list_set (struct ipi_vr *vr, char *name, int permit,
int preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route

`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_nexthop_prefix_list_unset

This function unsets the match next-hop address of a route. This function implements the `no match ip nexthop prefix list` command.

Syntax

```
int
route_map_match_ip_nexthop_prefix_list_unset (struct ipi_vr *vr, char *name, int
permit, int preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_nexthop_set

This function sets the match next-hop address of a route. This function implements the `match ip next-hop` command.

Syntax

```
int
route_map_match_ip_nexthop_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist.

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ip_nexthop_unset

This function unsets the match next-hop address of a route. This function implements the `no match ip next-hop` command.

Syntax

```
int
route_map_match_ip_nexthop_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:

`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_address_prefix_list_set

This function sets the match address of a route. This function implements the `match ipv6 address prefix-list` command.

Syntax

```
int
route_map_match_ipv6_address_prefix_list_set (struct ipi_vr *vr, char *name, int
permit, int preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_address_prefix_list_unset

This function unsets the match address of a route. This function implements the `no match ipv6 address prefix-list` command.

Syntax

```
int
route_map_match_ipv6_address_prefix_list_unset (struct ipi_vr *vr, char *name, int
permit, int preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_address_set

This function sets the match IPv6 address of a route. This function implements the `match ipv6 address` command.

Syntax

```
int
route_map_match_ipv6_address_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_address_unset

This function unsets the match IPv6 address of a route. This function implements the `no match ipv6 address` command.

Syntax

```
int
route_map_match_ipv6_address_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_nexthop_set

This function sets the match IPv6 next-hop address of a route. This function implements the `match ipv6 next-hop` command.

Syntax

```
int
route_map_match_ipv6_nexthop_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_ipv6_nexthop_unset

This function unsets the match IPv6 next-hop address of a route. This function implements the `no match ipv6 next-hop` command.

Syntax

```
int
route_map_match_ipv6_nexthop_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_metric_set

This function sets the match metric of a route. This function implements the `match metric` command.

Syntax

```
int
route_map_match_metric_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_metric_unset

This function unsets the match metric of a route. This function implements the `no match metric` command.

Syntax

```
int
route_map_match_metric_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_origin_set

This function sets the BGP origin code. This function implements the `match origin` command.

Syntax

```
int
route_map_match_origin_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_origin_unset

This function unsets the BGP origin code. This function implements the `no match origin` command.

Syntax

```
int
route_map_match_origin_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_route_type_set

This function sets the external route type. This function implements the `match route-type external` command.

Syntax

```
int
route_map_match_route_type_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_route_type_unset

This function unsets the external route type. This function implements the `no match route-type external` command.

Syntax

```
int
route_map_match_route_type_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_tag_set

This function sets a tag value. This function implements the `match tag` command.

Syntax

```
int
route_map_match_tag_set (struct ipi_vr *vr, char *name, int permit, int preference, char
*arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_match_tag_unset

This function unsets a tag value. This function implements the `no match tag` command.

Syntax

```
int
route_map_match_tag_unset (struct ipi_vr *vr, char *name, int permit, int preference,
char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_aggregator_as_set

This function sets the AS number of an aggregator. This function implements the `set aggregator as` command.

Syntax

```
int
route_map_set_aggregator_as_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_aggregator_as_unset

This function unsets the AS number of an aggregator. This function implements the `no set aggregator as` command.

Syntax

```
int
route_map_set_aggregator_as_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_as_path_prepend_set

This function sets the prepend string for a BGP AS-path attribute. This function implements the `set as-path prepend` command.

Syntax

```
int
route_map_set_as_path_prepend_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_as_path_prepend_unset

This function unsets the prepend string for a BGP AS-path attribute. This function implements the `no set as-path` command.

Syntax

```
int
route_map_set_as_path_prepend_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_atomic_aggregate_set

This function sets the BGP atomic aggregate attribute. This function implements the `set atomic-aggregate` command.

Syntax

```
int
route_map_set_atomic_aggregate_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_atomic_aggregate_unset

This function unsets the BGP atomic aggregate attribute. This function implements the `no set atomic-aggregate` command.

Syntax

```
int
route_map_set_atomic_aggregate_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_community_delete_set

This function sets the BGP community list for deletion. This function implements the `set comm-list` command.

Syntax

```
int
route_map_set_community_delete_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_community_delete_unset

This function unsets the BGP community list for deletion. This function implements the `no set comm-list` command.

Syntax

```
int
route_map_set_community_delete_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_community_set

This function sets the BGP community attribute. This function implements the `set community` command.

Syntax

```
int
route_map_set_community_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_community_unset

This function unsets the BGP community attribute. This function implements the `no set community` command.

Syntax

```
int
route_map_set_community_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_dampening_set

This function sets route-flap dampening. This function implements the `set dampening` command.

Syntax

```
int
route_map_set_dampening_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_dampening_unset

This function unsets route-flap dampening. This function implements the `no set dampening` command.

Syntax

```
int
route_map_set_dampening_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ext_community_rt_set

This function sets the route target extended community. This function implements the `set extcommunity rt` command.

Syntax

```
int
route_map_set_ext_community_rt_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ext_community_rt_unset

This function unsets the route target extended community. This function implements the `no set extcommunity rt` command.

Syntax

```
int
route_map_set_ext_community_rt_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ext_community_soo_set

This function sets the site-of-origin extended community. This function implements the `set extcommunity soo` command.

Syntax

```
int
route_map_set_ext_community_soo_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ext_community_soo_unset

This function unsets the site-of-origin extended community. This function implements the `no set extcommunity soo` command.

Syntax

```
int
route_map_set_ext_community_soo_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ip_nexthop_set

This function sets the IP address of the next hop. This function implements the `set ip next-hop` command.

Syntax

```
int
route_map_set_ip_nexthop_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ip_nexthop_unset

This function unsets the IP address of the next hop. This function implements the `no set ip next-hop` command.

Syntax

```
int
route_map_set_ip_nexthop_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ipv6_nexthop_local_set

This function sets the IPv6 next-hop address. This function implements the `set ipv6 next-hop local` command.

Syntax

```
int
route_map_set_ipv6_nexthop_local_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ipv6_nexthop_local_unset

This function unsets the IPv6 next-hop address. This function implements the `no set ipv6 next-hop local` command.

Syntax

```
int
route_map_set_ipv6_nexthop_local_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ipv6_nexthop_set

This function sets the IPv6 next-hop address. This function implements the `set ipv6 next-hop` command.

Syntax

```
int
route_map_set_ipv6_nexthop_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_ipv6_nexthop_unset

This function unsets the IPv6 next-hop address. This function implements the `no set ipv6 next-hop` command.

Syntax

```
int
route_map_set_ipv6_nexthop_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_local_preference_set

This function sets the BGP local preference path attribute. This function implements the `set local-preference` command.

Syntax

```
int
route_map_set_local_preference_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_local_preference_unset

This function unsets the BGP local preference path attribute. This function implements the `no set local-preference` command.

Syntax

```
int
route_map_set_local_preference_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_metric_set

This function sets the metric route-map. This function implements the `set metric` command.

Syntax

```
int
route_map_set_metric_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name

`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_metric_type_set

This function sets the type of metric for a destination routing protocol. This function implements the `set metric-type` command.

Syntax s

```
int
route_map_set_metric_type_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_metric_type_unset

This function unsets the type of metric for a destination routing protocol. This function implements the `no set metric-type` command.

Syntax

```
int
route_map_set_metric_type_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_metric_unset

This function unsets the metric value for a destination routing protocol. This function implements the `no set metric` command.

Syntax

```
int
route_map_set_metric_unset (struct ipi_vr *vr, char *name, int permit, int preference,
char * arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_originator_id_set

This function sets the BGP originator ID attribute. This function implements the `set originator-id` command.

Syntax

```
int
route_map_set_originator_id_set (struct ipi_vr *vr, char *name, int permit, int
preference, char *arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_originator_id_unset

This function unsets the BGP originator ID attribute. This function implements the `no set originator-id` command.

Syntax

```
int
route_map_set_originator_id_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_origin_set

This function sets the BGP origin code. This function implements the `set origin set` command.

Syntax

```
int
route_map_set_origin_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_origin_unset

This function unsets the BGP origin code. This function implements the `no set origin command`.

Syntax

```
int
route_map_set_origin_unset (struct ipi_vr *vr, char *name, int permit, int preference,
char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_tag_set

This function sets the tag value for a destination routing protocol. This function implements the `set tag` command.

Syntax

```
int
route_map_set_tag_set (struct ipi_vr *vr, char *name, int permit, int preference, char
*arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_tag_unset

This function unsets the tag value for a destination routing protocol. This function implements the `no set tag` command.

Syntax

```
int
route_map_set_tag_unset (struct ipi_vr *vr, char *name, int permit, int preference, char
* arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_vpnv4_nexthop_set

This function sets the VPNv4 next-hop address. This function implements the `set vpnv4-nexthop` command.

Syntax

```
int
route_map_set_vpnv4_nexthop_set (struct ipi_vr *vr, char *name,
                                  int permit, int pref, char *arg)
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_vpnv4_nexthop_unset

This function unsets the VPNv4 next-hop address. This function implements the `no set vpnv4-nexthop local` command.

Syntax

```
int
route_map_set_vpnv4_nexthop_local_unset (struct ipi_vr *vr, char *name, int permit, int
preference, char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds.

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_weight_set

This function sets the BGP weight for a routing table. This function implements the `set weight set` command.

Syntax

```
int
route_map_set_weight_set (struct ipi_vr *vr, char *name, int permit, int preference,
char *arg);
```

Input Parameters

`vr`	Virtual router

`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_COMPILE_ERROR when the compile function fails

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

route_map_set_weight_unset

This function sets the BGP weight for a routing table. This function implements the `no set weight` command.

Syntax

```
int
route_map_set_weight_unset (struct ipi_vr *vr, char *name, int permit, int preference,
char * arg);
```

Input Parameters

`vr`	Virtual router
`name`	Route-map name
`permit`	Whether to redistribute the route map; one of these constants from the `route_map_type` enum in `lib/routemap.h`:
`RMAP_PERMIT`	Redistribute the route
`RMAP_DENY`	Do not redistribute the route
`preference`	Preference or sequence number
`arg`	Command parameters

Output Parameters

None

Return Value

One (1) when a matching rule cannot be found

LIB_API_SET_SUCCESS when the function succeeds

LIB_API_SET_ERR_RMAP_RULE_MISSING when the route map rule is missing

LIB_API_SET_ERR_RMAP_NOT_EXIST when the route map does not exist

LIB_API_SET_ERR_RMAP_INDEX_NOT_EXIST when the route map index does not exist

CHAPTER 11 Logging API

This chapter describes the logging API for ZebOS-XP.

Directory Structure

Each subdirectory in the `pal` directory is an implementation of PAL for a specific operating system platform. The files in the `api` subdirectory are for baseline reference; the files in other directories provide implementation for them.

Logging APIs

This section describes the functions in the logging API.

Function	Description
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_close	Stops a log message

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`	Log file
`zl`	Module ID

Output Parameters

None

Return Values

Platform dependent

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`	Module instance
`zl`	Log instance
`dest`	Log destination

Return Values

Platform dependent

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 Parameter

`zg`	Log file
`zl`	Log module
`priority_str`	Priority of the message
`protocol`	Protocol or module
`message`	Buffer containing the data to log

Output Parameters

None

Return Values

Platform dependent

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

Platform dependent

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`	Log file

Output Parameters

None

Return Values

Platform dependent

Appendix A NSM Messages

This appendix introduces NSM messages and their components.

NSM Message Structures

NSM messaging protocol uses the type, length, value (TLV) notification. The CINDEX element of each message structure is set depending upon the types available from the message. The NSM_SET_CTYPE and NSM_CHECK_CTYPE macros are used to set and check the bits for the structure elements. Each message from the client has the following header:

```
#define NSM_DECODE_TLV_HEADER(TH)                                     \
    do {                                                              \
      TLV_DECODE_GETW ((TH).type);                                    \
      TLV_DECODE_GETW ((TH).length);                                  \
      (TH).length -= NSM_TLV_HEADER_SIZE;                             \
    } while (0)
#define NSM_CHECK_CTYPE(F,C)        (CHECK_FLAG (F, (1 << C)))
#define NSM_SET_CTYPE(F,C)          (SET_FLAG (F, (1 << C)))
#define NSM_CINDEX_SIZE                   32
typedef u_int32_t  cindex_t;
```

NSM Services

The following services are defined by NSM in `lib/nsm_message.h`.

Service	Identifier
Interface	`NSM_SERVICE_INTERFACE`
Router	`NSM_SERVICE_ROUTE`
Router ID	`NSM_SERVICE_ROUTER_ID`
Virtual Routing and Forwarding	`NSM_SERVICE_VRF`
Route Lookup	`NSM_SERVICE_ROUTE_LOOKUP`
Label Pool	`NSM_SERVICE_LABEL`
Traffic Engineering	`NSM_SERVICE_TE`
QOS	`NSM_SERVICE_QOS`
QoS Preemption from NSM	`NSM_SERVICE_QOS_PREEMPT`
User Management for Virtual Router	`NSM_SERVICE_USER`

Hostname (Obsolete)	`NSM_SERVICE_HOSTNAME`
MPLS VC	`NSM_SERVICE_MPLS_VC`
MPLS	`NSM_SERVICE_MPLS`
GMPLS	`NSM_SERVICE_GMPLS`
Diffserv	`NSM_SERVICE_DIFFSERV`
VPLS	`NSM_SERVICE_VPLS`
DS-TE	`NSM_SERVICE_DSTE`
IPv4 Multicast RIB	`NSM_SERVICE_IPV4_MRIB`
IPv4 PIM	`NSM_SERVICE_IPV4_PIM`
IPv4 Multicast Tunnel.	`NSM_SERVICE_IPV4_MCAST_TUNNEL`
IPv6 Multicast RIB	`NSM_SERVICE_IPV6_MRIB`
IPv6 PIM	`NSM_SERVICE_IPV6_PIM`
Bridge	`NSM_SERVICE_BRIDGE`
VLAN	`NSM_SERVICE_VLAN`
Interior Gateway Protocol Shortcut	`NSM_SERVICE_IGP_SHORTCUT`
Control Adjacency	`NSM_SERVICE_CONTROL_ADJ`
Control Channel	`NSM_SERVICE_CONTROL_CHANNEL`
Traffic Engineering Link	`NSM_SERVICE_TE_LINK`
Data Link	`NSM_SERVICE_DATA_LINK`
Data Link Subset	`NSM_SERVICE_DATA_LINK_SUB`
Global Identifier and Node Identifier	`SERVICE_GLOBAL_ID_NODE_ID`
Range limit	`NSM_SERVICE_MAX`

NSM Messages

Messages are defined in `lib/nsm_message.h`.

Message Constant	**Source**
`NSM_MSG_SERVICE_REQUEST`	Sent from protocol to request services from NSM
`NSM_MSG_SERVICE_REPLY`	Sent from NSM to protocol with a list of services available for a protocol

Message Constant	Source
`NSM_MSG_ADDR_ADD`	Sent from NSM to protocol when an interface address is added to NSM
`NSM_MSG_ADDR_DELETE`	Sent from NSM to protocol when an interface address is deleted from NSM
`NSM_MSG_ADDR_BULK_UPDATE`	(Reserved for future use)
`NSM_MSG_PROTOCOL_RESTART`	(Reserved for future use)
`NSM_MSG_USER_ADD`	Sent from NSM to protocol when a user is added to the system
`NSM_MSG_USER_UPDATE`	Sent from NSM to protocol when a password for a user is changed
`NSM_MSG_USER_DELETE`	Sent from NSM to protocol when a user is deleted from the system
`NSM_MSG_REDISTRIBUTE_SET`	Sent from protocol to NSM to set redistribution of a particular type of route from NSM
`NSM_MSG_REDISTRIBUTE_UNSET`	Sent from protocol to NSM to reset redistribution of a particular type of route from NSM
`NSM_MSG_ROUTER_ID_UPDATE`	Sent from NSM to protocol whenever a new Router ID is selected
`NSM_MSG_ADMIN_GROUP_UPDATE`	Sent from NSM to protocol whenever there is an administrative group update in NSM
`NSM_MSG_INTF_PRIORITY_BW_UPDATE`	Sent from NSM to protocol whenever priority bandwidth update message changes in NSM
`NSM_MSG_GMPLS_IF`	Sent from NSM to protocol with GMPLS configuration
`NSM_MSG_SUPPORTED_DSCP_UPDATE`	Sent from NSM to RSVP protocol to indicate that the supported DSCP information has changed in NSM
`NSM_MSG_DSCP_EXP_MAP_UPDATE`	Sent from NSM to RSVP protocol to indicate that the mapping between DSCP and EXP information has been changed in NSM
`NSM_MSG_DSTE_CT_UPDATE`	(Reserved for future use)
`NSM_MSG_DSTE_TE_CLASS_UPDATE`	Sent from NSM to protocol when DSTE TE Class is changed

Authentication Messages

Message Constant	Source
`NSM_MSG_AUTH_MAC_AUTH_STATUS`	Send MAC authentication status message to NSM
`NSM_MSG_AUTH_PORT_STATE`	Send port authentication state message to NSM
`NSM_MSG_MACAUTH_PORT_STATE`	Send MAC authentication port state message to NSM

CFM Messages

Message Constant	Source
`NSM_MSG_CFM_GET_IFINDEX`	Sent from NSM to protocol when an interface index changes in NSM
`NSM_MSG_CFM_OPERATIONAL`	Send "User Network Interface Maintenance Entity Group" CFM status to NSM
`NSM_MSG_CFM_REQ_IFINDEX`	Unused

ELMI Messages

Message Constant	Source
`NSM_MSG_ELMI_AUTO_VLAN_ADD_PORT`	Send ELMI auto VLAN config add port msg to NSM
`NSM_MSG_ELMI_AUTO_VLAN_DEL_PORT`	Send ELMI auto VLAN config delete port msg to NSM
`NSM_MSG_ELMI_EVC_BW_ADD`	Sent from NSM to protocol when bandwidth profile added per EVC
`NSM_MSG_ELMI_EVC_BW_DEL`	Sent from NSM to protocol when bandwidth profile deleted per EVC
`NSM_MSG_ELMI_EVC_COS_BW_ADD`	Sent from NSM to protocol when class of service added to the bandwidth
`NSM_MSG_ELMI_EVC_COS_BW_DEL`	Sent from NSM to protocol when class of service deleted from the bandwidth
`NSM_MSG_ELMI_EVC_DELETE`	Sent from NSM to protocol about EVC delete information
`NSM_MSG_ELMI_EVC_NEW`	Sent from NSM to protocol about EVC addition information
`NSM_MSG_ELMI_EVC_UPDATE`	Sent from NSM to protocol about EVC update information
`NSM_MSG_ELMI_OPERATIONAL_STATE`	Send ELMI operational status to NSM
`NSM_MSG_ELMI_UNI_ADD`	Sent from NSM to protocol about EVC UNI add message
`NSM_MSG_ELMI_UNI_BW_ADD`	Sent from NSM to protocol when BW profile parameters added per UNI
`NSM_MSG_ELMI_UNI_BW_DEL`	Sent from NSM to protocol when BW profile parameters deleted per UNI
`NSM_MSG_ELMI_UNI_DELETE`	Sent from NSM to protocol about the UNI delete information
`NSM_MSG_ELMI_UNI_UPDATE`	Sent from NSM to protocol about the UNI update information

G8031 and G8032 Messages

Message Constant	Source
`NSM_MSG_G8031_CREATE_PROTECTION_GROUP`	Sent from NSM to protocol about G8031 protection group add message
`NSM_MSG_G8031_CREATE_VLAN_GROUP`	Sent from NSM to protocol about G8031 VLAN group add message
`NSM_MSG_G8031_DEL_PROTECTION_GROUP`	Sent from NSM to protocol about G8031 protection group delete message
`NSM_MSG_G8031_PG_INITIALIZED`	Send protection group initialization msg to NSM
`NSM_MSG_G8031_PG_PORTSTATE`	Send protection group port state msg to NSM
`NSM_MSG_G8032_CREATE_VLAN_GROUP`	Sent from NSM to protocol about G80322 VLAN group add message

Interface Messages

Message Constant	Source
NSM_MSG_LINK_ADD	Sent from NSM to protocol when an interface is added to NSM
NSM_MSG_LINK_DELETE	Sent from NSM to protocol when an interface is deleted from NSM
NSM_MSG_LINK_BULK_UPDATE	(Reserved for future use)
NSM_MSG_LINK_ATTR_UPDATE	(Reserved for future use)
NSM_MSG_LINK_UP	Sent from NSM to protocol to indicate that an interface state has changed from down to up
NSM_MSG_LINK_DOWN	Sent from NSM to protocol to indicate that an interface state has changed from up to down

ISIS Messages

Message Constant	Source
`NSM_MSG_ISIS_BGP_CONV_DONE`	Sent from NSM to protocol to send BGP converged message to ISIS
`NSM_MSG_ISIS_BGP_DOWN`	Sent from NSM to protocol about the BGP down update
`NSM_MSG_ISIS_BGP_UP`	Sent from NSM to protocol about the BGP up update
`NSM_MSG_ISIS_WAIT_BGP_SET`	Sent from protocol to NSM to wait for BGP set message

Label Pool Messages

Message Constant	Source
NSM_MSG_LABEL_POOL_REQUEST	Sent from protocol to NSM to request a label block from NSM
NSM_MSG_LABEL_POOL_RELEASE	Sent from protocol to NSM to release a previously requested label pool block
NSM_MSG_GENERIC_LABEL_POOL_IN_USE	Sent from NSM to protocol about the generic Label pool is in use.
NSM_MSG_GENERIC_LABEL_POOL_RELEASE	Sent from NSM to protocol about the generic Label pool released.
NSM_MSG_GENERIC_LABEL_POOL_REQUEST	Sent from NSM to protocol about the generic Label pool request.

Layer-2 Messages

Message Constant	Source
NSM_MSG_BRIDGE_ADD	Sent from NSM to add a bridge to either STP, RSTP, or MSTP
NSM_MSG_BRIDGE_DELETE	Sent from NSM to delete a bridge from STP, RSTP, or MSTP
NSM_MSG_BRIDGE_ADD_PORT	Sent from NSM to add a switching port to a bridge
NSM_MSG_BRIDGE_DELETE_PORT	Sent from NSM to delete a switching port from a bridge
NSM_MSG_BRIDGE_ADD_G8032_RING	Sent from NSM to protocol when adding G8032 ring
NSM_MSG_BRIDGE_ADD_PG	Sent from NSM to protocol when adding STP to the protection group
NSM_MSG_BRIDGE_DEL_G8032_RING	Sent from NSM to protocol when deleting G8032 ring
NSM_MSG_BRIDGE_DEL_G8032_VLAN	Sent from NSM to protocol when deleting G8032 VLAN
NSM_MSG_BRIDGE_DEL_PG	Sent from NSM to protocol when deleting STP from the protection group
NSM_MSG_BRIDGE_DISABLE_AGEING	Send from MSTP to disable bridge aging
NSM_MSG_BRIDGE_G8032_PORT_STATE	Sent from protocols to NSM about the g8032 port state message
NSM_MSG_BRIDGE_PBB_TE_PORT_STATE	Sent from protocols to NSM about the PBB TE port state message
NSM_MSG_BRIDGE_PORT_FLUSH_FDB	Sent from NSM to protocols about the bridge port flush FDB message
NSM_MSG_BRIDGE_PORT_SPANNING_TREE_ENABLE	NSM sends to protocols about bridge interface message
NSM_MSG_BRIDGE_PORT_STATE	Sent from protocols to NSM about the bridge port state

Message Constant	Source
NSM_MSG_BRIDGE_PORT_STATE_SYNC_REQ	Sent from protocols to NSM about the bridge interface sync status
NSM_MSG_BRIDGE_SET_AGEING_TIME	Sent from protocols to NSM about the process bridge ageing time
NSM_MSG_BRIDGE_SET_STATE	Sent from protocols to NSM about the bridge set state message
NSM_MSG_BRIDGE_TCN	Send message to NSM about topology change detection
NSM_MSG_LACP_AGGREGATE_ADD	Sent from LACP to NSM for adding and updating the LACP aggregator
NSM_MSG_LACP_AGGREGATE_DEL	Sent from LACP to NSM for deleting an LACP aggregator
NSM_MSG_LACP_ADD_AGGREGATOR_MEMBER	Sent from NSM to protocols that a port has been aggregated
NSM_MSG_LACP_AGGREGATOR_CONFIG	Sent from NSM to protocols for configuration of LACP aggregator
NSM_MSG_LACP_DEL_AGGREGATOR_MEMBER	Sent from NSM to protocols that aggregated port has been deleted
NSM_MSG_STP_INTERFACE	Sent from protocols to NSM about the STP interface status
NSM_MSG_STATIC_AGG_CNT_UPDATE	Sent from NSM to LACP about adding or deleting count message of a static aggregator
NSM_MSG_UNTAGGED_VID_PE_PORT	Sent from NSM to protocols about the untagged VID for the PE port
NSM_MSG_SET_BPDU_PROCESS	NSM sends when configuring protocol handling on a customer edge or customer network port.

LDP Messages

Message Constant	Source
NSM_MSG_LDP_DOWN	NSM sends to ISIS when LDP service goes down
NSM_MSG_LDP_SESSION_QUERY	ISIS sends to NSM and NSM responds with LDP session information
NSM_MSG_LDP_SESSION_STATE	ISIS sends to NSM and NSM responds with LDP session information
NSM_MSG_LDP_UP	NSM sends to ISIS when LDP service comes up

Multicast Messages

Message Constant	Source
NSM_MSG_IGMP_JOIN	Sent from NSM to multicast routing protocol to indicate that a multicast group has `joined` on an interface
NSM_MSG_IGMP_LEAVE	Sent from NSM to multicast routing protocol to indicate that a multicast group has been `pruned` on an interface
NSM_MSG_IGMP_LMEM	Sent from NSM to protocol about IGMP local membership message
NSM_MSG_IPV4_VIF_ADD	Sent from multicast routing protocols to add a VIF in the MRIB
NSM_MSG_IPV4_VIF_DEL	Sent from multicast routing protocols to delete a VIF from the MRIB
NSM_MSG_IPV4_MRT_ADD	Sent from multicast routing protocols to add a multicast route in the MRIB
NSM_MSG_IPV4_MRT_DEL	Sent from multicast routing protocols to delete a multicast route from the MRIB
NSM_MSG_IPV4_MRT_STAT_FLAGS_UPDATE	Sent from multicast routing protocols to MRIB to change flags relating to forwarding statistics events for a multicast route
NSM_MSG_IPV4_MRT_NOCACHE	Sent from MRIB to multicast routing protocols when a NOCACHE event is received from the multicast forwarder
NSM_MSG_IPV4_MRT_WRONGVIF	Sent from MRIB to multicast routing protocols when a WRONGVIF event is received from the multicast forwarder
NSM_MSG_IPV4_MRT_WHOLEPKT_REQ	Sent from MRIB to multicast routing protocols when a WHOLEPKT event carrying a whole multicast data packet is received from multicast forwarder
NSM_MSG_IPV4_MRT_WHOLEPKT_REPLY	Sent from multicast routing protocols to MRIB in response to a WHOLEPKT request message
NSM_MSG_IPV4_MRT_STAT_UPDATE	Sent from MRIB to multicast routing protocols to indicate an immediate change in multicast route forwarding statistics or a timed statistics event notification
NSM_MSG_IPV4_MRIB_NOTIFICATION	A generic message from MRIB to multicast routing protocols carrying responses for VIF or MRT add and delete, notification to turn off or turn on multicast routing and multicast route clearing notification
NSM_MSG_IPV6_MRIB_NOTIFICATION	NSM sends MRIB notification to protocol
NSM_MSG_IPV4_MRT_ST_REFRESH_FLAG_UPDATE	NSM sends state refresh information updates to IGMP
NSM_MSG_IPV6_MIF_ADD	NSM sends multicast interface add information to protocol
NSM_MSG_IPV6_MIF_DEL	NSM sends multicast interface deletion updates to protocol.
NSM_MSG_IPV6_MRT_ADD	NSM sends multicast routing table add entry information to protocol
NSM_MSG_IPV6_MRT_DEL	NSM sends multicast routing table delete entry information to protocol
NSM_MSG_IPV6_MRT_NOCACHE	NSM sends multicast routing table nocache information to protocol

Message Constant	Source
NSM_MSG_IPV6_MRT_STAT_FLAGS_UPDATE	NSM sends multicast routing table state flag update information to protocol
NSM_MSG_IPV6_MRT_STAT_UPDATE	NSM sends multicast routing table state update to protocol
NSM_MSG_IPV6_MRT_ST_REFRESH_FLAG_UPDATE	NSM sends multicast routing table state refresh flag update to protocol module
NSM_MSG_IPV6_MRT_WHOLEPKT_REPLY	NSM sends multicast routing table whole packet reply to protocol
NSM_MSG_IPV6_MRT_WHOLEPKT_REQ	NSM sends multicast routing table whole packet request to protocol
NSM_MSG_IPV6_MRT_WRONGMIF	NSM sends multicast routing table wrong multicast interface updates to protocol
NSM_MSG_MLD_JOIN	Sent from MRIB to multicast routing protocols to indicate that a IPv6 multicast group has joined on an interface
NSM_MSG_MLD_LEAVE	Sent from MRIB to multicast routing protocols to indicate that a IPv6 multicast group has been pruned on an interface
NSM_MSG_MLD_LMEM	NSM sends MLD local- membership message to client

Nexthop Messages

Message Constant	Source
NSM_MSG_IPV4_NEXTHOP_BEST_LOOKUP	Sent from protocol to NSM to request a best-match route-lookup for an IPv4 route
NSM_MSG_IPV6_NEXTHOP_BEST_LOOKUP	Sent from protocol to NSM to request a best-match route-lookup for an IPv6 route
NSM_MSG_IPV4_NEXTHOP_EXACT_LOOKUP	Sent from protocol to NSM to request an exact-match route lookup for an IPv4 route
NSM_MSG_IPV6_NEXTHOP_EXACT_LOOKUP	Sent from protocol to NSM to request an exact-match route lookup for an IPv6 route
NSM_MSG_IPV4_NEXTHOP_BEST_LOOKUP_REG	Sent from protocol to NSM to register a best-match lookup for an IPv4 route
NSM_MSG_IPV4_NEXTHOP_BEST_LOOKUP_DEREG	Sent from protocol to NSM to deregister a best-match lookup for an IPv4 route
NSM_MSG_IPV4_NEXTHOP_EXACT_LOOKUP_REG	Sent from protocol to NSM to register an exact-match lookup for an IPv4 route
NSM_MSG_IPV4_NEXTHOP_EXACT_LOOKUP_DEREG	Sent from protocol to NSM to deregister an exact-match lookup for an IPv4 route
NSM_MSG_IPV6_NEXTHOP_BEST_LOOKUP_REG	Sent from protocol to NSM to register a best-match lookup for an IPv6 route

Message Constant	Source
NSM_MSG_IPV6_NEXTHOP_BEST_LOOKUP_DEREG	Sent from protocol to NSM to deregister a best-match lookup for an IPv6 route
NSM_MSG_IPV6_NEXTHOP_EXACT_LOOKUP_REG	Sent from protocol to NSM to register a exact-match lookup for an IPv6 route
NSM_MSG_IPV6_NEXTHOP_EXACT_LOOKUP_DEREG	Sent from protocol to NSM to deregister a exact-match lookup for an IPv6 route

OAM Messages

Message Constant	Description
NSM_MSG_OAM_CFM	NSM sends all CFM message opcode updates to protocol
NSM_MSG_OAM_LLDP	NSM sends all LLDP message updates to protocol
NSM_MSG_OAM_LSP_PING_REP_PROCESS_GEN	NSM sends to protocol about echo reply process message for Static LSP from oamd
NSM_MSG_OAM_LSP_PING_REP_PROCESS_L2VC	NSM sends to protocol about echo reply process message for L2VC type LSP from oamd
NSM_MSG_OAM_LSP_PING_REP_PROCESS_L3VPN	NSM sends to protocol about echo reply process message for L3VPN type LSP from oamd
NSM_MSG_OAM_LSP_PING_REP_PROCESS_LDP	NSM sends to protocol about echo reply process message for LDP type LSP from oamd
NSM_MSG_OAM_LSP_PING_REP_PROCESS_RSVP	NSM sends to protocol about echo reply process message for RSVP type LSP from oamd
NSM_MSG_OAM_LSP_PING_REP_RESP	NSM sends LSP ping report response message to protocol
NSM_MSG_OAM_LSP_PING_REQ_PROCESS	oamd sends echo request process message to NSM
NSM_MSG_OAM_LSP_PING_REQ_RESP_ERR	NSM sends LSP ping request response error message to sends.
NSM_MSG_OAM_LSP_PING_REQ_RESP_GEN	NSM sends LSP ping request response generic message to sends
NSM_MSG_OAM_LSP_PING_REQ_RESP_L2VC	NSM sends LSP ping request response L2VC message to protocol
NSM_MSG_OAM_LSP_PING_REQ_RESP_L3VPN	NSM sends LSP ping request response L3VPN message to sends
NSM_MSG_OAM_LSP_PING_REQ_RESP_LDP	NSM sends LSP ping request response message for LDP

Message Constant	Description
NSM_MSG_OAM_LSP_PING_REQ_RESP_RSVP	NSM sends LSP ping request response message for RSVP
NSM_MSG_OAM_UPDATE	NSM sends OAM update messages to protocol

PBB Messages

Message Constant	Source
NSM_MSG_PBB_DISPATCH	Unused
NSM_MSG_PBB_ISID_DEL	Unused
NSM_MSG_PBB_ISID_TO_BVID_ADD	NSM sends to protocols when setting an interface as a customer backbone port
NSM_MSG_PBB_ISID_TO_BVID_DEL	NSM sends to protocols when removing an interface as a customer backbone port
NSM_MSG_PBB_ISID_TO_PIP_ADD	NSM sends to protocols when adding a customer network port instance
NSM_MSG_PBB_SVID_TO_ISID_DEL	NSM sends to protocols when deleting a customer network port instance
NSM_MSG_PBB_TESID_INFO	NSM sends to protocols when adding a TE service instance
NSM_MSG_PBB_TE_APS_GRP_ADD	NSM sends to protocols when adding an automatic protection switching group
NSM_MSG_PBB_TE_APS_GRP_DELETE	NSM sends to protocols when deleting an automatic protection switching group
NSM_MSG_PBB_TE_ESP_ADD	NSM sends to protocols when adding an Ethernet Switched Path
NSM_MSG_PBB_TE_ESP_DELETE	NSM sends to protocols when deleting an Ethernet Switched Path
NSM_MSG_PBB_TE_ESP_PNP_ADD	NSM sends to protocols when a adding a Provider Network Port to an Ethernet Switched Path
NSM_MSG_PBB_TE_ESP_PNP_DELETE	NSM sends to protocols when a removing a Provider Network Port from an Ethernet Switched Path
NSM_MSG_PBB_TE_VID_ADD	NSM sends to protocols when adding a multicast Ethernet Switched Path
NSM_MSG_PBB_TE_VID_DELETE	NSM sends to protocols when deleting a multicast Ethernet Switched Path

PVLAN Messages

Message Constant	Source
NSM_MSG_PVLAN_CONFIGURE	NSM sends to protocols when adding a private VLAN
NSM_MSG_PVLAN_PORT_HOST_ASSOCIATE	NSM sends to protocols when making a layer2 port a host port or a promiscuous port.
NSM_MSG_PVLAN_SECONDARY_VLAN_ASSOCIATE	NSM sends to protocols when associating a secondary VLAN to a private VLAN

QoS Client Messages

Message Constant	Source
NSM_MSG_QOS_CLIENT_INIT	Sent from protocol to NSM to initiate the QoS module in NSM for that protocol
NSM_MSG_QOS_CLIENT_PROBE	Sent from protocol to NSM and from NSM to protocol. When sent from protocol to NSM, a QoS probe request. When sent from NSM to protocol, a response for a previously sent QoS probe request.
NSM_MSG_QOS_CLIENT_PROBE_RELEASE	Unused
NSM_MSG_QOS_CLIENT_RESERVE	Sent from protocol to NSM and from NSM to protocol. When sent from protocol to NSM, a QoS reserve request. When sent from NSM to protocol, a response for a previously sent QoS reserve request.
NSM_MSG_QOS_CLIENT_RELEASE	Sent from protocol to NSM to uninitiate the QoS module for a protocol
NSM_MSG_QOS_CLIENT_RELEASE_SLOW	Sent from protocol to NSM to release a QoS resource, based on all QoS parameters
NSM_MSG_QOS_CLIENT_PREEMPT	Sent from NSM to protocol whenever a QoS preempt occurs on a particular interface for a protocol
NSM_MSG_QOS_CLIENT_MODIFY	Sent from protocol to NSM and from NSM to protocol. When sent from protocol to NSM, a QoS modify request. When sent from NSM to protocol, a response for a previously sent QoS modify request.
NSM_MSG_QOS_CLIENT_CLEAN	Sent from protocol to NSM to clean up QoS resources for the protocol

Route Messages

Message Constant	Source
NSM_MSG_ROUTE_PRESERVE	Sent from protocol to NSM for graceful restart, so that protocol restarts and retains routes in NSM for a specified period
NSM_MSG_ROUTE_STALE_REMOVE	Sent from protocol to NSM to remove stale routes from NSM
NSM_MSG_ROUTE_IPV4	Sent from protocol to NSM when adding or deleting IPv4 routes. Also sent from NSM to protocol if a redistribution is requested by protocol or as a response to a route lookup.
NSM_MSG_ROUTE_IPV6	Sent from protocol to NSM when adding or deleting IPv6 routes. Also sent from NSM to protocol if a redistribution is requested by protocol or as a response to a route lookup.
NSM_MSG_ROUTE_CLEAN	(Reserved for future use)
`NSM_MSG_ROUTE_STALE_MARK`	Sent from protocol to NSM to mark stale routes

VLAN Messages

Message Constant	Source
`NSM_MSG_VLAN_ADD`	Sent from NSM to STP, RSTP or MSTP when adding a static VLAN; also sent from STP, RSTP or MSTP to NSM when adding a dynamic VLAN
`NSM_MSG_VLAN_DELETE`	Sent from NSM to STP, RSTP or MSTP when deleting a static VLAN; also sent from STP, RSTP or MSTP to NSM when deleting a dynamic VLAN
`NSM_MSG_VLAN_ADD_PORT`	Sent from NSM to STP, RSTP or MSTP when adding a VLAN to a port; also sent from STP, RSTP or MSTP to NSM when adding a VLAN to a port
`NSM_MSG_VLAN_DELETE_PORT`	Sent from NSM to STP, RSTP or MSTP when deleting a VLAN from a port; also sent from STP, RSTP or MSTP to NSM when deleting a VLAN from a port
`NSM_MSG_VLAN_PORT_TYPE`	Sent by NSM to STP, RSTP or MSTP to set the port type
`NSM_MSG_VLAN_CLASSIFIER_ADD`	Sent from NSM to STP, RSTP and MSTP when adding a VLAN classifier
`NSM_MSG_VLAN_CLASSIFIER_DEL`	Sent from NSM to STP, RSTP and MSTP when deleting a VLAN classifier
`NSM_MSG_VLAN_PORT_CLASS_ADD`	Sent from NSM to STP, RSTP and MSTP when setting a VLAN classifier group on an access port
`NSM_MSG_VLAN_PORT_CLASS_DEL`	Sent from NSM to STP, RSTP and MSTP when clearing a VLAN classifier group from an access port
`NSM_MSG_VLAN_ADD_TO_INST`	Protocol sends to NSM about the VLAN add from a instance in a bridge
`NSM_MSG_VLAN_ADD_TO_PROTECTION`	Protocol sends to NSM about the VLAN add updates from the spanning-tree
`NSM_MSG_VLAN_DELETE_FROM_INST`	Protocol sends to NSM about the VLAN delete from a instance in a bridge

Message Constant	Source
NSM_MSG_VLAN_DEL_FROM_PROTECTION	Protocol sends to NSM about the VLAN delete updates from the spanning-tree
NSM_MSG_VLAN_SET_PVID	NSM sends to protocol for the PVID updates

VR Messages

Message Constant	Source
NSM_MSG_VR_ADD	Sent from NSM to protocol when a Virtual Router is added
NSM_MSG_VR_DELETE	Sent from NSM to protocol when a Virtual Router is deleted
NSM_MSG_VR_UPDATE	(Reserved for future use)
NSM_MSG_VR_BIND	Sent from NSM to protocol when an interface is bound to a Virtual Router
NSM_MSG_VR_BIND_BULK	(Reserved for future use)
NSM_MSG_VR_UNBIND	Sent from NSM to protocol when an interface is unbound from a Virtual Router
NSM_MSG_VR_SYNC_DONE	NSM sends VR sync done message to protocol

VRF Messages

Message Constant	Source
NSM_MSG_VRF_ADD	Sent from NSM to protocol when a VRF is added
NSM_VRF_DELETE	Sent from NSM to protocol when a VRF is deleted
NSM_MSG_VRF_UPDATE	(Reserved for future use)
NSM_MSG_VRF_BIND	Sent from NSM to protocol when an interface is bound to a VRF
NSM_MSG_VRF_BIND_BULK	(Reserved for future use)
NSM_MSG_VRF_UNBIND	Sent from NSM to protocol when an interface is unbound from a VRF

Miscellaneous Messages

Message Constant	Source
`NSM_MSG_BGP_CONV_DONE`	Protocol sends BGP converged done message to NSM
`NSM_MSG_BLOCK_INSTANCE`	MSTP sends to protection groups in NSM marking the instance as blocked for protection
`NSM_MSG_UNBLOCK_INSTANCE`	MSTP sends to NSM to unmark the instance so that it can be used for protection groups
`NSM_MSG_COMMSG`	Protocol sends the COMMSG updates to NSM
`NSM_MSG_DEFAULT_VID_PE_PORT`	NSM receives default VID PE port information from protocol
`NSM_MSG_DLINK_CHANNEL_MONITOR`	Unused
`NSM_MSG_DLINK_OPAQUE`	LMP sends to NSM to make data links opaque or transparent
`NSM_MSG_DLINK_SEND_TEST_MESSAGE`	Client sends dlink test message to NSM
`NSM_MSG_EFM_OAM_IF`	EFM sends interface status message to NSM
`NSM_MSG_IF_DEL_DONE`	Clients send to NSM to notify that delete processing is complete
`NSM_MSG_LMP_GRACEFUL_RESTART`	Client sends about LMP graceful restart message to NSM
`NSM_MSG_MRTR_PRESENT`	NSM send to protocols to indicate that an IGMP multicast router is present
`NSM_MSG_MTRACE_QUERY`	NSM sends to protocols
`NSM_MSG_MTRACE_REQUEST`	NSM sends to protocols
`NSM_MSG_NEXTHOP_TRACKING`	Protocol sends nexthop tracking information to NSM
`NSM_MSG_NSM_SERVER_STATUS`	NSM sends to protocols when shutting down
`NSM_MSG_RMON_REQ_STATS`	Protocol sends RMON request for statistics to NSM
`NSM_MSG_RMON_SERVICE_STAT`	Protocol sends remote monitoring service statistics to NSM
`NSM_MSG_RSVP_CONTROL_PACKET`	NSM sends RSVP control packet to protocol
`NSM_MSG_SVLAN_ADD_CE_PORT`	Protocol sends SVLAN customer edge port add message to NSM
`NSM_MSG_SVLAN_DELETE_CE_PORT`	Protocol sends SVLAN customer edge port delete message to NSM

Index

A

Address Resolution Protocol APIs 99
APIs
  Address Resolution Protocol 99
  ARP 99
  Interface Evolution MIB 47, 49
  logging 151
  NSM SNMP MIB 95
  Route Map 103
ARP APIs 99

B

bandwidth probe
  QRM 87

F

FIB 65
Forwarding Information Base 65
Forwarding Information Base interface 65

G

global bandwidth management
  QRM 87

H

hardware interface management 35

I

Interface Evolution MIB APIs 47, 49
Interface Management
  overview 35
interface messages 159

L

label pool messages 160
Layer-2 messages 160
logging APIs 151
LSP resource reservation
  QRM 87

M

message handler 12
multicast messages 162
multiple preemption
  QRM 87

N

Network Services Module
  Quality of Service Resource Manager 87
nexthop messages 163
NSM client 12
NSM client disconnect handling 12
NSM client parser and callback 12
NSM messages 156
  interface messages 159
  label pool messages 160
  Layer-2 messages 160
  multicast messages 162
  nexthop messages 163
  QOS client messages 166
  route messages 167
  VLAN messages 167
  VR messages 168
  VRF messages 168
NSM server 12
NSM services 155
NSM SNMP MIB APIs 95
nsm_api_arp_entry_add 99
nsm_api_arp_entry_del 100
nsm_get_rcvaddress_status 47
nsm_get_rcvaddress_type 48
nsm_get_sys_desc 96
nsm_get_sys_location 96
nsm_get_sys_name 96
nsm_get_sys_oid 97
nsm_get_sys_services 97
nsm_get_sys_up_time 98
nsm_if_proxy_arp_set 100
nsm_qos_serv_clean_for 88
nsm_qos_serv_if_deinit 89
nsm_qos_serv_if_get 89
nsm_qos_serv_if_init 89
nsm_qos_serv_init 90
nsm_qos_serv_update_max_bw 90
nsm_qos_serv_update_max_resv_bw 91
nsm_read_qos_client_clean 91
nsm_read_qos_client_init 92
nsm_read_qos_client_modify 92
nsm_read_qos_client_probe 92
nsm_read_qos_client_release 93
nsm_read_qos_client_reserve 93
NSM-Protocol messaging protocol 155

P

pal_if_mip6_home_agent_set 67
pal_if_mip6_home_agent_unset 67
pal_kernel_fib_create 67
pal_kernel_fib_delete 68
pal_kernel_gratuitous_arp_send 68
pal_kernel_if_flags_get 69
pal_kernel_if_flags_set 69
pal_kernel_if_flags_unset 70
pal_kernel_if_get_bw 70
pal_kernel_if_get_hwaddr 70
pal_kernel_if_get_index 71
pal_kernel_if_get_metric 71
pal_kernel_if_get_mtu 71
pal_kernel_if_info 72
pal_kernel_if_ipv4_address_add 72
pal_kernel_if_ipv4_address_delete 72
pal_kernel_if_ipv4_address_delete_all 73
pal_kernel_if_ipv4_address_secondary_add 73
pal_kernel_if_ipv4_address_secondary_delete 74
pal_kernel_if_ipv4_address_update 74
pal_kernel_if_ipv6_address_add 74
pal_kernel_if_ipv6_address_delete 75
pal_kernel_if_scan 75
pal_kernel_if_unbind_vrf 76
pal_kernel_if_update 76
pal_kernel_ipv4_add 76
pal_kernel_ipv4_del 77
pal_kernel_ipv4_forwarding_get 77
pal_kernel_ipv4_forwarding_set 77
pal_kernel_ipv4_update 78
pal_kernel_ipv6_add 78
pal_kernel_ipv6_del 79
pal_kernel_ipv6_forwarding_get 79
pal_kernel_ipv6_forwarding_set 79
pal_kernel_ipv6_old_del 80
pal_kernel_ipv6_update 80
pal_kernel_L2_ipv4_resolve 81
pal_kernel_L2_ipv6_resolve 81
pal_kernel_route_scan 81
pal_kernel_start 74
pal_kernel_stop 82
pal_kernel_virtual_ipv4_add 83
pal_kernel_virtual_ipv4_delete 83
pal_kernel_virtual_mac_add 83
pal_kernel_virtual_mac_delete 84
pal_kernel_vrrp_start 84
pal_log_close 151
pal_log_open 152
pal_log_output 152
pal_log_start 152
pal_log_stop 153
preemption priorities 87

Q

QOS client messages 166
QRM, see Quality of Service Resource Manager
Quality of Service (QoS) module interface 87
Quality of Service Resource Manager 87
  bandwidth probe support 87
  global bandwidth management 87
  LSP resource reservation 87
  multiple preemption 87
  resource preemption 87

R

resource preemption
  QRM 87
Route Map APIs 103
route messages 167
route_map_match_as_path_set 106
route_map_match_as_path_unset 106
route_map_match_community_set 107
route_map_match_community_unset 108
route_map_match_interface_set 108
route_map_match_interface_unset 109
route_map_match_ip_address_prefix_list_set 110
route_map_match_ip_address_prefix_list_unset 110
route_map_match_ip_address_set 111
route_map_match_ip_address_unset 112
route_map_match_ip_nexthop_prefix_list_set 112
route_map_match_ip_nexthop_prefix_list_unset 113
route_map_match_ip_nexthop_set 114
route_map_match_ip_nexthop_unset 114
route_map_match_ipv6_address_prefix_list_set 115
route_map_match_ipv6_address_prefix_list_unset 116
route_map_match_ipv6_address_set 116
route_map_match_ipv6_address_unset 117
route_map_match_ipv6_nexthop_set 118
route_map_match_ipv6_nexthop_unset 118
route_map_match_metric_set 119
route_map_match_metric_unset 120
route_map_match_origin_set 120
route_map_match_origin_unset 121
route_map_match_route_type_set 122
route_map_match_route_type_unset 122
route_map_match_tag_set 123
route_map_match_tag_unset 124
route_map_set_aggregator_as_set 124
route_map_set_aggregator_as_unset 125
route_map_set_as_path_prepend_set 126
route_map_set_as_path_prepend_unset 126
route_map_set_atomic_aggregate_set 127
route_map_set_atomic_aggregate_unset 128
route_map_set_community_delete_set 128
route_map_set_community_delete_unset 129
route_map_set_community_set 130
route_map_set_community_unset 130
route_map_set_dampening_set 131
route_map_set_dampening_unset 132
route_map_set_ext_community_rt_set 132
route_map_set_ext_community_rt_unset 133
route_map_set_ext_community_soo_set 134
route_map_set_ext_community_soo_unset 134
route_map_set_ip_nexthop_set 135

route_map_set_ip_nexthop_unset 136
route_map_set_ipv6_nexthop_local_set 136
route_map_set_ipv6_nexthop_local_unset 137
route_map_set_ipv6_nexthop_set 138
route_map_set_ipv6_nexthop_unset 138
route_map_set_local_preference_set 139
route_map_set_local_preference_unset 140
route_map_set_metric_set 140
route_map_set_metric_type_set 141
route_map_set_metric_type_unset 142
route_map_set_metric_unset 142
route_map_set_origin_set 144
route_map_set_origin_unset 145
route_map_set_originator_id_set 143
route_map_set_originator_id_unset 144
route_map_set_tag_set 146
route_map_set_tag_unset 146
route_map_set_vpnv4_nexthop_set 147
route_map_set_vpnv4_nexthop_unset 148
route_map_set_weight_set 148
route_map_set_weight_unset 149

S

system interface information 35

V

VLAN messages 167
VR messages 168
VRF messages 168