ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Layer 2 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

This guide describes the ZebOS-XP application programming interface (API) for Layer 2 protocols.

Audience

This guide is intended for developers who write code to customize and extend Layer 2 protocols.

Conventions

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

Table P-1: Conventions

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

Contents

This guide contains these chapters:

- Chapter 1, *Overview of ZebOS-XP Layer 2 Networking*
- Chapter 2, *Multiple Spanning Tree Protocol*
- Chapter 3, *Rapid Per-VLAN Spanning Tree Plus*
- Chapter 4, *Flow Control*
- Chapter 5, *Layer 2 Gateway Port*
- Chapter 6, *Port Mirroring*
- Chapter 7, *Virtual LANs*
- Chapter 8, *VLAN Registration Protocol*
- Chapter 9, *Port Authentication*
- Chapter 10, *Link Aggregation*
- Chapter 11, *Remote Monitoring*

Related Documents

The following guides are related to this document:

- *Layer 2 Command Reference*
- *Layer 2 Configuration Guide*
- *Carrier Ethernet Developer Guide*
- *Carrier Ethernet Command Reference*
- *Carrier Ethernet Configuration Guide*
- *Data Center Bridging Developer Guide*
- *Data Center Bridging Command Reference*
- *Data Center Bridging Configuration Guide*
- *Ethernet Local Management Interface Developer Guide*
- *Ethernet Local Management Interface Command Reference*
- *Ethernet Local Management Interface Configuration Guide*
- *Transparent Interconnection of Lots of Links Developer Guide*
- *Transparent Interconnection of Lots of Links Configuration Guide*
- *Transparent Interconnection of Lots of Links Command Reference*
- *Precision Time Protocol Developer Guide*
- *Precision Time Protocol Configuration Guide*
- *Precision Time Protocol Command Reference*
- *Synchronous Ethernet Developer Guide*
- *Synchronous Ethernet Configuration Guide*
- *Synchronous Ethernet Command Reference*
- *Shortest Path Bridging Developer Guide*
- *Shortest Path Bridging Configuration Guide*
- *Shortest Path Bridging Command Reference*
- *Edge Virtual Bridging Developer Guide*
- *Edge Virtual Bridging Configuration Guide*
- *Edge Virtual Bridging Command Reference*

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 of ZebOS-XP Layer 2 Networking

This chapter introduces the Layer 2 protocol modules.

ZebOS-XP includes these Layer 2 features which are covered in this document:

- Spanning Tree
- Virtual Local Area Networks
- Registration Frameworks
- Port Authentication (authd)
- Link Aggregation
- Multi-Chassis Link Aggregation
- Remote Monitoring

Figure 1-1 shows the ZebOS-XP Layer 2 high-level architecture.

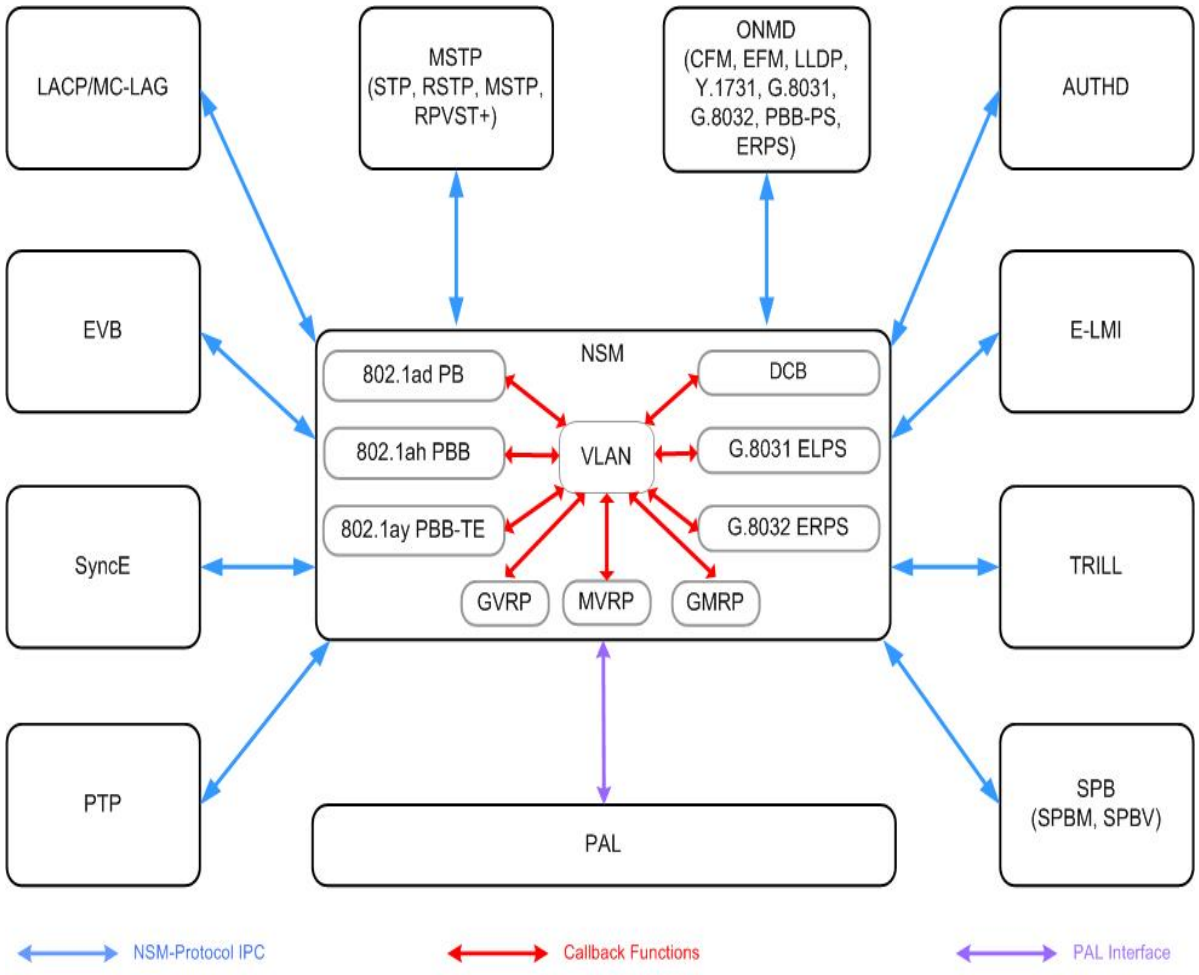

Figure 1-1: Layer 2 architecture

Note: Not all the modules shown in Figure 1-1 are described in this document. See Other Layer 2 Modules for more.

Spanning Tree

The ZebOS-XP Spanning Tree modules are a combination of these modules:

- Spanning Tree Protocol (STP)
- Rapid Spanning Tree Protocol (RSTP)
- Multiple Spanning Tree Protocol (MSTP)

The following highlights the features of the Spanning Tree Protocol modules.

Note: All ZebOS-XP spanning tree modules support 802.3x flow control, broadcast storm recovery, and port mirroring.

Spanning Tree Protocol (802.1d)

The ZebOS-XP Spanning Tree Protocol (STP) module creates spanning trees within mesh networks of Layer 2 connected bridges, disabling any links that are not a part of the tree and leaving a single active connection between any two unique network nodes.

STP devices exchange BPDU (bridge protocol data unit) messages. The Spanning Tree Algorithm calculates the best path and prevents multiple paths between network segments. STP elects a root bridge, finds paths and determines the least cost path to the root bridge, then disables all other paths.

Network managers may design a topology that uses redundant links as automatic backup paths in the case of active link failure. Automatic backup takes place without the pitfalls of bridge loops, or the need to manually enable or disable backup links.

ZebOS-XP STP supports all STP switch port states, including:

- Listening, learning, blocking, forwarding, disabled

For more information about the spanning tree modules, see:

- Chapter 2, *Multiple Spanning Tree Protocol*
- Chapter 3, *Rapid Per-VLAN Spanning Tree Plus*
- Chapter 4, *Flow Control*
- Chapter 5, *Layer 2 Gateway Port*
- Chapter 6, *Port Mirroring*

Rapid Spanning Tree Protocol (802.1w)

The Rapid Spanning Tree Protocol (RSTP) accelerates the re-configuration and restoration of a spanning tree after a link failure.

Multiple Spanning Tree Protocol (802.1s)

The Multiple Spanning Tree Protocol (MSTP) is a supplement to the IEEE 802.1ad standard. MSTP allows VLAN bridges to use multiple spanning trees, by providing the ability for traffic belonging to different VLANs to flow over potentially different paths within the virtual bridged LAN.

Virtual Local Area Networks

The VLAN modules offer a consistent network-wide management tools to manage virtual LANs (Local Area Networks) and bridged VLANs:

- VLAN bridging allows network devices to segment into VLANs, regardless of their physical location.
- VLANs, in accordance with IEEE 802.1Q, enable multiple bridged LANs to transparently share the same physical network link without leaking information between LANs. Traffic between VLANs is restricted to bridges that forward unicast, multicast, or broadcast traffic only on the LAN segments that serve the VLAN to which the traffic belongs.

ZebOS-XP VLAN modules make it easy to administer logical groups of stations that can communicate as if they were on the same LAN. They make it easier to manage a move, add, delete, or other updates to members of these groups.

The ZebOS-XP VLAN modules support all IEEE 802.1D LAN MAC (Media Access Control) protocols, shared media, and point-to-point LANs. MAC bridging allows multiple LANs to be connected together. MAC bridging filters data sent between LAN segments, reduces network congestion, and allows networks to be partitioned for administrative purposes.

For more information about VLANs, see Chapter 7, *Virtual LANs*.

Registration Frameworks

The Layer 2 support in ZebOS-XP includes these registration frameworks:

- Generic Attribute Registration Protocol: GARP is a generic framework for bridges to register and de-register attributes, such as VLAN identifiers and multicast group membership.
- GARP VLAN Registration Protocol (802.1Q): GVRP is a GARP application that provides VLAN registration service. GVRP uses GARP Information Declaration (GID) and GARP Information Propagation (GIP), which provide the common state machine descriptions and the common information propagation mechanisms defined for GARP-based applications.

  GVRP provides support for VLAN pruning and dynamic VLAN creation. A switch can exchange VLAN configuration information with other GVRP switches, prune unnecessary broadcast and unknown unicast traffic, and dynamically create and manage VLANs.
- GARP Multicast Registration Protocol: GMRP allows participants to dynamically register and de-register information with the Media Access Control (MAC) bridges attached to the same LAN segment. A switch can exchange multicast group information with other GMRP switches, prune unnecessary broadcast traffic, and dynamically create and manage multicast groups.
- Multicast Registration Protocol (802.1ak): MRP has protocols, procedures, and managed objects to support multiple registrations, and allow the participants in an MRP application to register attributes with other participants in a bridged LAN.
- Multiple VLAN Registration Protocol (802.1ak): MVRP registers multiple VLANs and provides for the rapid healing of network failures without interrupting services to unaffected VLANs. In addition, MVRP improves the convergence time of the GVRP module.
- Multiple Multicast Registration Protocol (802.1ak) MMRP manages group Media Access Control (MAC) addresses. In addition, MMRP improves the convergence time of GMRP.

For more about the registration frameworks, see Chapter 8, *VLAN Registration Protocol*.

Port Authentication (authd)

The ZebOS-XP Layer 2 802.1x module provides port-based network access control for LAN devices. The IEEE 802.1x standard offers centralized control of user authentication and access. For more, see Chapter 9, *Port Authentication*.

Link Aggregation

The Link Aggregation module allows one or more links to be aggregated together to form a Link Aggregation Group (LAG), such that a MAC client can treat the Link Aggregation Group as if it were a single link. The Link Aggregation Control Protocol (LACP) allows bundling of several physical interfaces to form a single logical channel providing enhanced performance and redundancy. The aggregated interface is viewed as a single link to each switch. The spanning tree also views it as one interface. When there is a failure in one physical interface, the remaining interfaces stay up, so there is no disruption. For more, see Chapter 10, *Link Aggregation*.

Multi-Chassis Link Aggregation

Multi-Chassis Link Aggregation (also called MC-LAG, MLAG, or Distributed Resilient Network Interconnect [DRNI]) extends the link aggregation concept to ensure that connectivity between two networks can be maintained despite the failure of a node. With MC-LAG, at either one or both ends of a link aggregation group, a single aggregation system is replaced by a *portal* that is a collection of one to three portal systems. For more, see Chapter 10, *Link Aggregation*.

Remote Monitoring

Remote monitoring (RMON; defined by RFC RFC 2819) provides remote monitoring and management of network devices and enables these devices to communicate with each other to exchange network information. For more see Chapter 11, *Remote Monitoring*.

Other Layer 2 Modules

Table 1-1 shows supported Layer 2 modules that are described in other ZebOS-XP documents.

Table 1-1: ZebOS-XP Layer 2 modules described in other documents

Layer 2 feature	Document
Career Ethernet, including: Provider Bridging (PB), Provider Backbone Bridging (PBB) Provider Backbone Bridging-Traffic Engineering (PBB-TE) Ethernet Linear Protection Switching (ELPS) Ethernet Ring Protection Switching (ERPS) Connectivity Fault Management (CFM) Ethernet in the First Mile (EFM) Link Layer Discovery Protocol (LLDP)	*Carrier Ethernet Developer Guide*
Data Center Bridging (DCB)	*Data Center Bridge Developer Guide*

Table 1-1: ZebOS-XP Layer 2 modules described in other documents

Layer 2 feature	Document
Ethernet Local Management Interface (E-LMI)	*Ethernet Local Management Interface Developer Guide*
TRILL (TRansparent Interconnection of Lots of Links)	*Transparent Interconnection of Lots of Links Developer Guide*
Precision Time Protocol (PTP)	*Precision Time Protocol Developer Guide*
Synchronous Ethernet (SyncE)	*Synchronous Ethernet Developer Guide*
Shortest Path Bridging (SPB)	*Shortest Path Bridging Developer Guide*
Edge Virtual Bridging (EVB)	*Edge Virtual Bridging Developer Guide*

CHAPTER 2 Multiple Spanning Tree Protocol

The ZebOS-XP Multiple Spanning Tree Protocol (MSTP) module enables devices to avoid bridge loops by exchanging BPDU (bridge protocol data unit) messages. In addition, MSTP provides Management Information Base (MIB) support. MSTP calculates the best path to help prevent multiple paths between network segments.

Overview

ZebOS-XP MSTP creates spanning trees within mesh networks of Layer 2 connected bridges and disables any links that are not a part of the tree, which leaves a single active connection between any two unique network nodes. MSTP selects a root bridge and finds the correct path to determine the least cost path to the root bridge. It then disables all other paths. Users can design a topology that uses redundant links as automatic backup paths in the case of active link failure. Automatic backup takes place without the pitfalls of bridge loops or the need to manually enable or disable backup links.

Supported STP Modules

ZebOS-XP supports the following spanning tree protocol modules.

Spanning Tree Protocol (802.1d)

The Spanning Tree Protocol (STP) module enables devices to avoid bridge loops by exchanging BPDU (bridge protocol data unit) messages. The ZebOS-XP implementation supports IEEE 802.1D and RFC 4188.

Multiple Spanning Tree Protocol (802.1s)

Multiple Spanning Tree Protocol (MSTP) is a supplement to the IEEE 802.1Q standard. It allows VLAN bridges to use multiple spanning trees by allowing traffic belonging to different VLANs to flow over potentially different paths within the virtual bridged LAN. The ZebOS-XP MSTP module is also optimized for fast convergence times to support the MSTP MIB requirements in draft-malhotra-mstpmib-01.txt.

Rapid Spanning Tree Protocol (802.1w)

The Rapid Spanning Tree Protocol (RSTP) accelerates reconfiguring and restoraing a spanning tree after a link failure. The ZebOS-XP implementation of RSTP optimizes convergence time, and supports both IEEE 802.1D and RFC 4318.

Rapid per VLAN Spanning Tree Plus

The Rapid per VLAN Spanning Tree Plus (RPVST+) protocol builds an individual spanning-tree topology for each VLAN defined on a bridge. This topology runs on RPVST+ and 802.1q trunks. When a client puts a switch in RPVST+ mode, the switch can then support both RPVST+ and 802.1q inter-switch trunks. In RPVST+ mode, each VLAN runs its own independent spanning-tree instance. In addition, RPVST+ bridges can have different spanning-tree topologies for different VLANs within an autonomous switching domain.

See Chapter 3, *Rapid Per-VLAN Spanning Tree Plus* for more information on RPVST+.

Flow Control (802.3x)

Flow control is the process of managing the rate of data transmission between two nodes. Flow control allows a sending computer to transmit information at a faster rate than the destination computer can receive and process them.

This can happen if the receiving computer has a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

See Chapter 4, *Flow Control* for more information on flow control.

Layer 2 Gateway Protocol

The Layer 2 Gateway Protocol (L2GP) solution helps separate different Spanning Tree Protocols (STP) domains. In this solution, the user configures one or more ports as layer-two gateway ports, with each port defining the border of a domain in which the STP algorithm is active. The L2GP solution implements STP as a hello protocol and defines an L2GP as a regular port.

See Chapter 5, *Layer 2 Gateway Port* for more information on L2GP.

Port Mirroring

Port mirroring is used on network switches to send copies of all network packets seen on one switch port to a network monitoring connection on another switch port. This is commonly used for network appliances that require monitoring of network traffic, such as an intrusion-detection system. ZebOS-XP Spanning Tree protocol modules support port mirroring.

See Chapter 6, *Port Mirroring* for more information on port mirroring.

Data Structures

The following are the common data structures that are used with all MSTP protocol modules.

mstp_bridge

This data structure is defined in mstpd/mstp_types.h.

Member	Definition
`next`	Housekeeping variables (of type mstp_bridge)
`pprev`	Housekeeping variables (of type mstp_bridge)
`port_list`	MSTP Port related information (of type mstp_port)
`vlan_table`	VLAN tree indexed on VID (of type route_table)
`config`	Configuration information (of type mstp_config_info)
`name`	Bridge name, size (L2_BRIDGE_NAME_LEN +1)

Member	Definition
type	Bridge type whose value are NSM_BRIDGE_TYPE_STP NSM_BRIDGE_TYPE_STP_VLANAWARE NSM_BRIDGE_TYPE_RSTP NSM_BRIDGE_TYPE_RSTP_VLANAWARE NSM_BRIDGE_TYPE_MSTP NSM_BRIDGE_TYPE_PROVIDER_RSTP NSM_BRIDGE_TYPE_PROVIDER_MSTP NSM_BRIDGE_TYPE_CE NSM_BRIDGE_TYPE_RPVST_PLUS NSM_BRIDGE_TYPE_BACKBONE_RSTP NSM_BRIDGE_TYPE_BACKBONE_MSTP
vlan_list, fid_list	List of VLANs added to the common instance (of type rlist_info).
low_port	Index of the lowest numbered port (by index)
bridge_addr	Bridge's MAC address (of type mac_addr)
cist_bridge_id	Cist Bridge id (of type bridge_id)
cist_designated_root	Designated root of type bridge -id (CIST Root priority vectors)
cist_reg_root	Reg root of type bridge_id (CIST Root priority vectors)
cist_rcvd_reg_root	Received reg root of type bridge_id (Cist Root priority vectors)
cist_designated_bridge	Designated bridge of type bridge_type (CIST Root priority vectors)
external_root_path_cost	External root path cost (CIST Root priority vectors)
internal_root_path_cost	Internal root path cost (CIST Root priority vectors)
cist_root_port_id	Root Port id (CIST Root priority vectors)
cist_root_port_ifindex	Roof Port interface index (CIST Root priority vectors)
bpdu_recv_port_ifindex	Receive Port interface index (CIST Root priority vectors)
root_port	Root port (CIST Root priority vectors)
alternate_port	Alternate port (CIST Root priority vectors)
cist_max_age	Max age (Working values - CIST root times)
cist_message_age	Message age (Working values - CIST root times)
cist_hello_time	Hello time (Working values - CIST root times)
cist_forward_delay	Forward delay (Working values - CIST root times)
hop_count	Hop count (Working values - CIST root times)

Member	Definition
force_version	Force version (Working values - CIST root times)
bridge_max_age	Bridge Max age (Configuration values - Bridge times)
bridge_hello_time	Bridge hello time (Configuration values - Bridge times)
bridge_forward_delay	Bridge forward delay (Configuration values - Bridge times)
bridge_max_hops	Bridge max hops (Configuration values - Bridge times)
cist_bridge_priority	Bridge priority (Configuration values - Bridge times)
recent_root	Recent root (Configuration values - Bridge times)
br_all_rr_timer_cnt	Recent root timer count
ageing_time	How long dynamic entries are kept (Default 300secs)
learning_enabled	BPDU Flag
Mstp_enabled	BPDU Flag
mstp_brforward	BPDU Flag
topology_change	BPDU Flag
topology_change_detected	BPDU flag
bridge_enabled	BPDU flag
is_vlan_aware	BPDU flag.
reselect	BPDU flag
ageing_time_is_fwd_delay	Flag to represent whether ageing time is forward delay time or not
ce_bridge	Flag to represent whether bridge is CE
cn_bridge	Flag to represent whether bridge is CN
is_default	Flag to represent whether bridge is default
ha_stale	Flag to represent HA stale
path_cost_method	Used for path cost method
port_index	Port index
instance_list	Hold RPVST or MST bridge instance (of type mstp_bridge_instance).
max_instances	Maximum instances whose values RPVST_MAX_INSTANCES (if rpvst) MST_MAX_INSTANCES (if mst)
num_ports	Number of ports

Member	Definition
time_last_topo_change	Holds statistics of last topology change
l2gp_pathcost_control	L2GP path-cost control
num_topo_changes	Number of topology changes
total_num_topo_changes	Total number of topology changes
max_age_count	Max age count
bpduguard	Layer 2 security related parameters
errdisable_timeout_interval	Layer 2 security related parameters
errdisable_timeout_enable	Layer 2 security related parameters
bpdu_filter	Layer 2 security related parameters
oper_cisco	Layer 2 security related parameters
admin_cisco	Layer 2 security related parameters
transmit_hold_count	Layer 2 security related parameters
topology_type	Topology type whose values are MSTP_TOPOLOGY_NONE and MSTP_TOPOLOGY_RING
topology_change_timer	Topology change timer (of type thread)
tcn_timer	TCN timer (of type thread)
tc_initiator	Statistics of CIST level
tc_flag	Statistics for CIST level
tc_last_rcvd_from	Statistics of CIST level
vlan_instance_map	RPVST VLAN instance map
cust_bpdu_process	Processing customer BPDU on customer edge port
mstp_bridge_cdr_ref	CDR Reference for MSTP bridge
topology_change_timer_cdr_ref	CDR Reference for topology change timer
tcn_timer_cdr_ref	CDR Reference for TCN timer
mstpProtectionBmp	MSTP Protection BMP (of type mstp_protection_bmp)

Definition

```
struct mstp_bridge
{
  /* Housekeeping variables */
```

```
  struct mstp_bridge *          next;
  struct mstp_bridge * *        pprev;
  struct mstp_port *            port_list;

  /* VLAN tree indexed on vid. */
  struct route_table            *vlan_table;

  struct mstp_config_info       config;

  char                          name[L2_BRIDGE_NAME_LEN+1];

  u_int8_t                      type;

 /* List of vlans added to the common instance */
  struct rlist_info *           vlan_list;
  struct rlist_info *           fid_list;

  /* Index of the lowest numbered port (by ifindex) */
  s_int32_t                     low_port;

  /* Since the priority is diff for each instance we just store the mac addr
   * i.e. the addr of the lowest indexed port from which bridge id
   * is calculated by each instance */
  struct mac_addr               bridge_addr;
  struct bridge_id              cist_bridge_id;

  /* MSTP -CIST root priority vector sec 13.23.3 IEEE 802.1s */
  struct bridge_id              cist_designated_root;
  struct bridge_id              cist_reg_root;
  struct bridge_id              cist_rcvd_reg_root;
 u_int32_t                     external_root_path_cost;
  u_int32_t                     internal_root_path_cost;
  u_int16_t                     cist_root_port_id;
  u_int32_t                     cist_root_port_ifindex;
  u_int32_t                     bpdu_recv_port_ifindex;
  struct mstp_port*             root_port;
  struct mstp_port*             alternate_port;

  /* Working values - CIST root times sec 17.17.7 */
  s_int32_t                     cist_max_age;
  s_int32_t                     cist_message_age;
  s_int32_t                     cist_hello_time;
  s_int32_t                     cist_forward_delay;
  s_int32_t                     hop_count;
  u_char                        force_version;
  /* Configuration values- bridge times sec 17.17.4  */
  s_int32_t                     bridge_max_age;
  s_int32_t                     bridge_hello_time;
  s_int32_t                     bridge_forward_delay;
  s_int32_t                     bridge_max_hops;

  u_int32_t                     cist_bridge_priority;

  u_int32_t                     recent_root;

  /* Recent Root Timer Count */
  u_int32_t                     br_all_rr_timer_cnt;
```

```
  /* How long dynamic entries are kept. Default 300s */
  s_int32_t                     ageing_time;

  u_char                        learning_enabled;
  /* BPDU flags - see 802.1d */
  u_char                        mstp_enabled:1;
  u_char                        mstp_brforward:1;
  u_char                        topology_change:1;
  u_char                        topology_change_detected:1;
  u_char                        bridge_enabled:1;
  u_char                        is_vlan_aware:1;
  u_char                        reselect:1;

u_char                        ageing_time_is_fwd_delay:1;
  u_char                        ce_bridge:1;
#if (defined HAVE_I_BEB)
  u_char                        cn_bridge:1;
#endif

  u_char                        is_default:1;
#ifdef HAVE_HA
  u_char                        ha_stale:1;
#endif /* HAVE_HA */

  /* Used for Path cost method Short/Long */
  u_int8_t                      path_cost_method;

  s_int16_t                     port_index;

#ifdef HAVE_RPVST_PLUS
  struct mstp_bridge_instance   *instance_list[RPVST_MAX_INSTANCES];
#else
  struct mstp_bridge_instance   *instance_list[MST_MAX_INSTANCES];
#endif /* HAVE_RPVST_PLUS */

  u_int16_t max_instances;

  s_int16_t                     num_ports;

  /* Statistics */
  pal_time_t   time_last_topo_change;
  u_int32_t    num_topo_changes;
  u_int32_t    total_num_topo_changes;
  u_int32_t    max_age_count;

 /* L2 security related parameters */
  unsigned char                 bpduguard;
  s_int32_t                     errdisable_timeout_interval;
  unsigned char                 errdisable_timeout_enable;
  unsigned char                 bpdu_filter;
  unsigned char                 oper_cisco;
  unsigned char                 admin_cisco;
  unsigned char                 transmit_hold_count;
/* Topology type - none/ring - currently used to support RRSTP */
  enum mstp_topology topology_type;
```

```
  struct thread *               topology_change_timer;
  struct thread *               tcn_timer;

 struct thread *             hold_timer;

  /* Statistics */
  u_int32_t                   cist_forward_transitions;

  /* VLAN Membership of the port. Is CVLAN is case of.
   * CE port and is SVLAN in the case of CN and PN port
   */
  struct mstp_vlan_bmp        vlanMemberBmp;

  /* SVLAN Membership of the CE port. This is used to
   * keep track of the logical PE port
   */
  struct mstp_vlan_bmp        svlanMemberBmp;

  /* SVLAN ID of the Provider Edge Port */

  u_int16_t                   svid;

  u_int8_t                    spanning_tree_disable;

#if (defined HAVE_I_BEB )
  struct mstp_bridge         *cn_br;

  uint32_t                    isid;

  uint32_t                    pip_port; /* if_index */

  uint32_t                    bvid;

  struct list                 * svid_list;

#endif


  /* only used for rpvst_plus */
#ifdef HAVE_RPVST_PLUS
  /* Hello Timer fired or Rcvd superior bpdu */
  enum rpvst_bpdu_event       rpvst_event;

  /* BPDU type is 8021.D(CIST)  or SSTP(for each vlan) */
  enum rpvst_bpdu_type        rpvst_bpdu_type;
/* Set this flag when new bpdu for a vlan to be tx */
  bool_t                    newInfoSstp;

  /* Vid to be used in 802.1Q tag for SSTP Bpdus */
  u_int16_t                   vid_tag;
  int                       default_vlan;
#endif /* HAVE_RPVST_PLUS */


#ifdef HAVE_HA
HA_CDR_REF mstp_instance_port_cdr_ref; /* CDR ref for mstp_instance_port */
HA_CDR_REF errdisable_timer_cdr_ref; /* CDR ref for errdisable timer */
```

```
  HA_CDR_REF port_timer_cdr_ref; /* CDR ref for port_timer */
  HA_CDR_REF edge_delay_timer_cdr_ref; /* CDR ref for edge_delay timer */
HA_CDR_REF forward_delay_timer_cdr_ref; /* CDR ref for forward_delay timer
*/
HA_CDR_REF hello_timer_cdr_ref; /* CDR ref for hello timer */
  HA_CDR_REF migrate_timer_cdr_ref; /* CDR ref for migrate timer */
HA_CDR_REF recent_backup_timer_cdr_ref; /* CDR ref for recent_backup timer
*/
HA_CDR_REF message_age_timer_cdr_ref; /* CDR ref for message_age timer */
HA_CDR_REF recent_root_timer_cdr_ref; /* CDR ref for recent_root timer */
HA_CDR_REF tc_timer_cdr_ref; /* CDR ref for tc timer */
HA_CDR_REF hold_timer_cdr_ref; /* CDR ref for hold timer */
#endif /* HAVE_HA */
/* statistics variables */
u_int32_t conf_bpdu_sent;
u_int32_t conf_bpdu_rcvd;
u_int32_t tcn_bpdu_sent;
u_int32_t tcn_bpdu_rcvd;
u_int32_t src_mac_count;
u_int32_t similar_bpdu_cnt;
u_int32_t msg_age_timer_cnt;
 u_int32_t total_src_mac_count;

  /* BPDU Counters */
  ut_int64_t bpdu_discards_rx;
  ut_int64_t bpdu_discards_tx;
  ut_int64_t bpdu_guard_events;
  ut_int64_t enableBPDUrx_discards_count;
  ut_int64_t enableBPDUtx_discards_count;
};
```

mstp_bridge_instance

This data structure is defined in mstpd/mstp_types.h:

Member	Definition
next	Housekeeping variable
pprev	Housekeeping variable
bridge	Housekeeping variable
port_list	Each instance has its own port list. The IST has all ports added to it
vlan_list	List of VLANs added to the common instance (of type rlist_info)
fid_list	List of VLANS added to the common instance (of type rlist_info)
instance_id	Instance identifier
vlan_range_index_bmp	Bitmap for VLAN range index, used by SNMP
low_port	Index of the lowest numbered port (by ifindex)

Member	Definition
master	Indicates if the bridge has selected one of its ports as a master port
learning_enabled	Indicates if the bridge has selected one of its ports as a master port
msti_mastered	Indicates if the bridge has selected one of its ports as a master port
reselect	Indicates if the bridge has selected one of its ports as a master port
mstp_enabled	Indicates if the bridge has selected one of its ports as a master port
msti_bridge_id	Indicates if the bridge has selected one of its ports as a master port
msti_bridge_priority	Indicates if the bridge has selected one of its ports as a master port
recent_root	Index of the port requesting other recent roots to revert to discarding state
br_inst_all_rr_timer_cnt	Recent root timer count
msti_designated_root	MSTP - MST root priority vector
msti_designated_bridge	MSTP - MST root priority vector
internal_root_path_cost	MSTP - MST root priority vector
msti_root_port_id	MSTP - MST root priority vector
root_inst_port	MSTP - MST root priority vector.
msti_root_port_ifindex	MSTP - MST root priority vector
port_index	Port index
hop_count	Hop count
tc_flag	Statistics for instance/VLAN level
topology_change_detected	Statistics for instance/VLAN level
time_last_topo_change	Statistics for instance/VLAN level
num_topo_changes	Statistics for instance/VLAN level
total_num_topo_changes	Statistics for instance/VLAN level
tc_initiator	Statistics for instance/VLAN level
tc_last_rcvd_from	Statistics for instance/VLAN level
vlan_id	MSTI port times variable
message_age	MSTI port times variable
max_age	MSTI port times variable

Member	Definition
fwd_delay	MSTI port times variable
hello_time	MSTI port times variable
mstp_bridge_instance_cdr_ref	CDR ref for mstp_bridge_instance
is_te_instance	Flag to represent the “te” instance

Definition

```
 /* Housekeeping variables */
  struct mstp_bridge_instance *              next;
  struct mstp_bridge_instance * *            pprev;
  struct mstp_bridge *                       bridge;

  /* Each instance has its own port list
   * the IST has all ports added to it */
  struct mstp_instance_port *           port_list;

  struct rlist_info *            vlan_list;
  struct rlist_info *            fid_list;

  char                           instance_id;

  /* Bitmap for Vlan Range Index, used by snmp */
  struct bitmap *vlan_range_index_bmp;

  /* Index of the lowest numbered port (by ifindex) */
  s_int32_t                      low_port;

  /* indicates if the bridge has selected one of its ports
   * as master ports */
  u_char                        master:1 ;
  u_char                        learning_enabled:1;
  u_char                        msti_mastered:1;
  u_char                        reselect:1;
  /*L2-R3 */
#if defined (HAVE_PROVIDER_BRIDGE) || defined (HAVE_B_BEB)
  u_char                        mstp_enabled:1;
#endif /*(HAVE_PROVIDER_BRIDGE) ||(HAVE_B_BEB)*/

  struct bridge_id              msti_bridge_id;
  u_int32_t                     msti_bridge_priority;

  /* Index of the port requesting other recent roots to revert to
     discarding state */
  u_int32_t                     recent_root;

  /* Recent Root Timer Count */
  u_int32_t                     br_inst_all_rr_timer_cnt;

  /* MSTP -MST root priority vector sec 17.4.2.2 and sec 17.17.6*/
  struct bridge_id              msti_designated_root;
```

```
  struct bridge_id              msti_designated_bridge;
  u_int32_t                     internal_root_path_cost;
  u_int16_t                     msti_root_port_id;
  struct mstp_instance_port *   root_inst_port;
  u_int32_t                     msti_root_port_ifindex;

  s_int16_t                     port_index;

  s_int32_t                     hop_count;

   /* statistics for instance/vlan level */
  bool_t                        tc_flag;
  bool_t                        topology_change_detected;
  pal_time_t                    time_last_topo_change;
  u_int32_t                     num_topo_changes;
  u_int32_t                     total_num_topo_changes;
  u_int16_t                     tc_initiator;
  u_char                        tc_last_rcvd_from[ETHER_ADDR_LEN];

#ifdef HAVE_RPVST_PLUS
  u_int16_t                     vlan_id;
  /* MSTI Port times variable */
  s_int32_t                    message_age;
  s_int32_t                    max_age;
  s_int32_t                    fwd_delay;
  s_int32_t                    hello_time;
#endif /* HAVE_RPVST_PLUS */

#ifdef HAVE_HA
  HA_CDR_REF mstp_bridge_instance_cdr_ref; /* CDR ref for mstp_bridge_instance
*/
#endif /* HAVE_HA */

  s_int32_t is_te_instance;

};
```

mstp_port

The API data structure for the following objects resides in the mstpd/mstp_types.h file.

Member	Definition
next	House Keeping variables
pprev	House Keeping variables
instance_list	Bridge instance list (of type mstp_instance_port)
br	Bridge information (of type mstp_bridge)
ce_br	CE Bridge information (of type mstp_bridge)
dev_addr	Net Device address of size ETHER_ADDR_LEN

Member	Definition
ifindex	Interface index
name	Interface name of size L2_IF_NAME_LEN
tx_count	Outgoing BPDU counter
total_tx_count	Number of outgoing BPDUs
total_rx_count	Number of incoming BPDUs
force_version	Force version
info_internal	Flag representing internal message
rcvd_internal	Flag representing external message
admin_p2p_mac	Flag representing admin link type
oper_p2p_mac	Flag representing operational link type
admin_edge	Flag representing admin edge
oper_edge	Flag representing operational edge
auto_edge	Flag representing auto edge
portfast_conf	Flag representing portfast configuration
port_enabled	Flag representing port is enabled or not
rcvd_mstp	Flag representing valid BPDUs received
rcvd_stp	Flag representing valid BPDUs received
rcvd_rstp	Flag representing valid BPDUs received
send_mstp	Flag representing valid BPDUs sent
tc_ack:1	Flag representing topology change acknowledgement
selected	Flag representing port is selected or not
reselect	Flag representing port is reselected or not
send_proposal	Flag representing send proposal
rcv_proposal	Flag representing receive proposal
updtInfo	Flag representing updated information
pathcost_configured	Flag representing path cost is configured
disputed	Flag representing port is disputed
agree	Flag representing port is agreed

Member	Definition
agreed	Flag representing port is agreed
config_bpdu_pending	Flag representing configuration BPDUs are pending
ha_stale	Flag representing HA stale
isL2gp	Flag representing is port is L2GP
enableBPDUrx	Flag representing port is enabled for receiving BPDUs
enableBPDUtx	Flag representing port is enabled to send BPDUs
bridge_id	Bridge identifier
psuedoRootId	Pseudo rood identifier
admin_bpduguard	Admin BPDU guard (L2 security information)
oper_bpduguard	Operational BPDU guard (L2 Security information)
errdisable_timeout_enable	BPDU guard timeout status
errdisable_timeout_interval	BPDU guard timeout interval
errdisable_timer	Error disable timer (of type thread)
admin_bpdufilter	Flag representing ports admin BPDU filter is enabled
oper_bpdufilter	Flag representing ports operational BPDU filter is enabled
admin_rootguard	Flag representing ports admin root guard is enabled.
oper_rootguard	Flag representing ports operational root guard is enabled.
cisco_cfg_format_id	Cisco configuration format identifier.
admin_loopguard	Flag representing ports admin loop guard enabled.
oper_loopguard	Flag representing ports operational loop guard enabled.
restricted_role	Flag representing ports restricted role is enabled.
restricted_tcn	Flag representing ports restricted tcn is enabled.
hello_time	Hold ports hello time.
ref_count	Hold ports reference count.
type	Port type whose value is either TYPE_EXPLICIT or TYPE_IMPLICIT.

Member	Definition
port_type	Ports port type whose values include: • ACCESS_PORT • HYBRID_PORT • TRUNK_PORT • CUST_EDGE_PORT • CUST_NET_PORT • PRO_NET_PORT • CUSTOMER_NETWORK_PROVIDER_PORT • PROVIDER_INSTANCE_PORT
any_msti_rootport	Any MSTI root port
any_msti_desigport	Any MSTI designated port
cist_path_cost	CIST path cost
cist_priority	CIST priority
cist_port_id	Combo of ifindex and priority
cist_root	CIST Root (CSTI port priority vector)
cist_reg_root	CIST Reg Root (CSTI port priority vector)
cist_rcvd_reg_root	CIST Received Reg Root (CSTI port priority vector)
cist_designated_bridge	CIST Designated bridge (CSTI port priority vector)
cist_external_rpc	CIST External Root Path Cost (CSTI port priority vector)
cist_internal_rpc	CIST Internal Root Path Cost (CSTI port priority vector)
cist_designated_port_id	CIST designated port identifier (CSTI port priority vector)
cist_message_age	CIST message age (CSTI Port times variable)
cist_max_age	CIST max age (CSTI Port times variables)
cist_fwd_delay	CIST Forward Delay (CSTI Port times variable)
cist_hello_time	CIST Hello time (CSTI Port times variable).
hop_count	Hop count (CSTI Port times variable)
newInfoCist	Flag represent ports newinfocist is enabled
newInfoMsti	Flag representing ports newinfomsti is enabled
critical_bpdu	Flag representing ports critical BPDU is enabled
cist_state	CIST state whose values include: • STATE_DISCARDING • STATE_LISTENING • STATE_LEARNING • STATE_FORWARDING • STATE_BLOCKING

Member	Definition
cist_next_state	CIST Port next state (of type port_state)
ist_role	Cist Port role whose values include one of the following: • ROLE_MASTERPORT • ROLE_ALTERNATE • ROLE_ROOTPORT • ROLE_DESIGNATED • ROLE_DISABLED • ROLE_BACKUP
cist_selected_role	CIST Port selected role (of type port_role).
cist_tc_state	CIST Port TC state whose values include one of the following: • TC_INACTIVE • TC_ACTIVE
helloWhen	Flag representing port hello when is enabled
ort_timer	Port timer
edge_delay_timer	edgeDelayWhile timer
forward_delay_timer	fdWhile timer
hello_timer	helloWhen timer
migrate_timer	mdelayWhile timer
recent_backup_timer;	rbWhile timer.
message_age_timer	rcvdInfoWhile.
recent_root_timer	rrWhile timer.
tc_timer	tcWhile time.
hold_timer	Hold timer.
cist_forward_transitions	CIST forward transitions statistics.
vlanMemberBmp	VLAN membership of the port. CVLAN is case of CE port and is SVLAN in the case of CN and PN ports.
svlanMemberBmp	SVLAN membership of a CE port. This is used to keep track of the logical PE port.
svid	SVLAN ID of the provider edge port.
spanning_tree_disable	Flag represent ports spanning tree disable is enabled or not.
cn_br	CN Bridge info (of type mstp_bridge).
isidpip	
port	

Member	Definition
bvid	Bridged VLAN ID
svid_list	Service VLAN ID list
rpvst_event	RPVST BPDU event whose values include one of the following: • RPVST_PLUS_TIMER_EVENT • RPVST_PLUS_RCVD_EVENT
rpvst_bpdu_type	RPVST BPDU type, including: • RPVST_PLUS_BPDU_TYPE_CIST • RPVST_PLUS_BPDU_TYPE_UNTAGGED • RPVST_PLUS_BPDU_TYPE_TAGGED • RPVST_PLUS_BPDU_TYPE_SSTP
newInfoSstp	Set this flag when new BPDU for a VLAN needs to be transmitted
vid_tag	VID to be used in 802.1Q tag for SSTP BPDUs
default_vlan	Default VLAN
mstp_port_cdr_ref	CDR reference for MSTP port
errdisable_timer_cdr_ref	CDR reference for err-disable timer
port_timer_cdr_ref	CDR reference for port timer.
edge_delay_timer_cdr_ref	CDR reference for edge delay timer
forward_delay_timer_cdr_ref	CDR reference for forward delay timer
hello_timer_cdr_ref	CDR reference for hello timer
migrate_timer_cdr_ref	CDR Reference for migrate timer
recent_backup_timer_cdr_ref	CDR Reference for recent backup timer
message_age_timer_cdr_ref	CDR Reference for message age timer
recent_root_timer_cdr_ref	CDR Reference for recent root timer
tc_timer_cdr_ref	CDR Reference for TC timer
hold_timer_cdr_ref	CDR Reference for hold timer
conf_bpdu_sent	Configuration BPDU Sent (statistics)
conf_bpdu_rcvd	Configuration BPDU Received (statistics)
tcn_bpdu_sent	TCN BPDU sent (statistics)
tcn_bpdu_rcvd	TCN BPDU Received (statistics)
src_mac_count	Source MAC count (statistics)
similar_bpdu_cnt	Similar BPDU count (statistics)

Member	Definition
msg_age_timer_cnt	Message age timer count.
total_src_mac_count	Total source MAC count
bpdu_discards_rx	BPDU discards Receive counter
bpdu_discards_tx	BPDU discards transmit counter
bpdu_guard_events	BPDU guard events
enableBPDUrx_discards_count	Enable BPDU receive discard counter
enableBPDUtx_discards_count	Enable BPDU transmit counter

Definition

```
struct mstp_port
{
  /* Housekeeping */
  struct mstp_port *          next;
  struct mstp_port **         pprev;

  struct mstp_instance_port *      instance_list;
  struct mstp_bridge *             br;

  struct mstp_bridge *             ce_br;

  u_char                      dev_addr[ETHER_ADDR_LEN];
  u_int32_t                   ifindex;
  char                        name[L2_IF_NAME_LEN];

  u_char                      tx_count;
  /* Number of outgoing BPDUs not discarded by BPDU Filter or due to disabled
BPDU tx */
  ut_int64_t                  total_tx_count;
  /* Number of incoming BPDUs not discarded by BPDU Guard/Filter or due to
disabled BPDU rx */
  ut_int64_t                  total_rx_count;

  u_char                      force_version;

  u_char                      info_internal:1;
  u_char                      rcvd_internal:1;

  u_char                      admin_p2p_mac:2;
  u_char                      oper_p2p_mac:1;
  u_char                      admin_edge:1;
  u_char                      oper_edge:1;
  u_char                      auto_edge:1;

  u_char                      portfast_conf:1;
  u_char                      port_enabled:1;
  u_char                      rcvd_mstp:1;
  u_char                      rcvd_stp:1;
  u_char                      rcvd_rstp:1;
```

```
  u_char                      send_mstp:1;
  u_char                      tc_ack:1;

 u_char                      selected:1;
  u_char                      send_proposal:1;
  u_char                      rcv_proposal:1;
  u_char                      reselect:1; /* Section 17.18.29 */
  u_char                      updtInfo:1;
  u_char                      pathcost_configured:1;
  u_char                      disputed:1;
  u_char                      agree:1;
  u_char                      agreed:1;
  u_char                      config_bpdu_pending:1;
#ifdef HAVE_HA
  u_char                      ha_stale:1;
#endif /* HAVE_HA */

  u_char                      enableBPDUrx; /* 802.1ah-d4-1 13.25.18 */
  u_int8_t                    enableBPDUtx; /* 802.1ah-d4-1 13.25.19 */
#ifdef HAVE_L2GP
  u_char                      isL2gp; /* 802.1ah-d4-1 13.25.21 */
  struct bridge_id            psuedoRootId; /* 802.1ah-d4-1 13.25.20 */
#endif /* HAVE_L2GP */

  /* L2 security related information */
  s_int32_t                   admin_bpduguard;
  u_char                      oper_bpduguard;
  /* BPDU Guard timeout status */
  u_int8_t                    errdisable_timeout_enable;
  /* BPDU Guard timeout interval in tics */
  u_int32_t                   errdisable_timeout_interval;
  struct thread *             errdisable_timer;
  u_char                      admin_bpdufilter;
  u_char                      oper_bpdufilter;
  u_char                      admin_rootguard;
  u_char                      oper_rootguard;
  u_char                      cisco_cfg_format_id;

  /* Loop Guard administrative and operational states */
  u_int8_t                    admin_loopguard;
  u_int8_t                    oper_loopguard;
 bool_t                      restricted_role;
  bool_t                      restricted_tcn;
  s_int32_t                   hello_time;

  char                        ref_count;
  enum add_type               type;
  char                        port_type;
  char                        any_msti_rootport;
  char                        any_msti_desigport;

  u_int32_t                   cist_path_cost; /*  */
  s_int16_t                   cist_priority;
  u_int16_t                   cist_port_id;  /* Combo of ifindex and priority
*/

  /* CSTI  Port Priority Vector */
```

```
  struct bridge_id            cist_root;
  struct bridge_id            cist_reg_root;
  struct bridge_id            cist_rcvd_reg_root;
  struct bridge_id            cist_designated_bridge;
  u_int32_t                   cist_external_rpc;
  u_int32_t                   cist_internal_rpc;
  s_int32_t                   cist_designated_port_id;

  /* CSTI  Port times variable */
  s_int32_t                    cist_message_age;
  s_int32_t                    cist_max_age;
  s_int32_t                    cist_fwd_delay;
  s_int32_t                    cist_hello_time;
  s_int32_t                    hop_count;

  bool_t                       newInfoCist;
  bool_t                       newInfoMsti;
  bool_t                       critical_bpdu;
 enum port_state             cist_state;
  enum port_state             cist_next_state;
  enum info_type              cist_info_type; /* not needed probably */
  enum port_role              cist_role;
  enum port_role              cist_selected_role;
  enum tc_state               cist_tc_state;

  /* Flags */
  /* State Machine Variables */

  u_char                      helloWhen;

  /* port timer  */
  struct thread *             port_timer;

  /* edgeDelayWhile timer Section 17.17.1 */
  struct thread *             edge_delay_timer;

  /* fdWhile timer Section 17.17.2 */
  struct thread *             forward_delay_timer;

  /* helloWhen timer Section 17.17.3 */
  struct thread *             hello_timer;

  /* mdelayWhile Timer Section 17.17.4 */
  struct thread *             migrate_timer;

  /* rbWhile timer Section 17.17.5 */
  struct thread *             recent_backup_timer;

  /* rcvdInfoWhile Section 17.17.6 */
  struct thread *             message_age_timer;

  /* rrWhile timer Section 17.17.7 */
  struct thread *             recent_root_timer;
 /* tcWhile timer Section 17.17.8 */
  struct thread *             tc_timer;

struct thread *             hold_timer;
```

```
  /* Statistics */
  u_int32_t                   cist_forward_transitions;

  /* VLAN Membership of the port. Is CVLAN is case of.
   * CE port and is SVLAN in the case of CN and PN port
   */
  struct mstp_vlan_bmp        vlanMemberBmp;

  /* SVLAN Membership of the CE port. This is used to
   * keep track of the logical PE port
   */
  struct mstp_vlan_bmp        svlanMemberBmp;

  /* SVLAN ID of the Provider Edge Port */

  u_int16_t                   svid;

  u_int8_t                    spanning_tree_disable;

#if (defined HAVE_I_BEB )
  struct mstp_bridge         *cn_br;

  uint32_t                    isid;

  uint32_t                    pip_port; /* if_index */
  uint32_t                    bvid;

  struct list                 * svid_list;

#endif

  /* only used for rpvst_plus */
#ifdef HAVE_RPVST_PLUS
  /* Hello Timer fired or Rcvd superior bpdu */
  enum rpvst_bpdu_event       rpvst_event;

  /* BPDU type is 8021.D(CIST)  or SSTP(for each vlan) */
  enum rpvst_bpdu_type        rpvst_bpdu_type;
/* Set this flag when new bpdu for a vlan to be tx */
  bool_t                    newInfoSstp;

  /* Vid to be used in 802.1Q tag for SSTP Bpdus */
  u_int16_t                   vid_tag;
  int                       default_vlan;
#endif /* HAVE_RPVST_PLUS */

#ifdef HAVE_RPVST_PLUS
u_chartc_ack:1;
bool_tnewInfoSstp;
charany_msti_rootport;
charany_msti_desigport;
u_int16_tvlan_id;
#endif /* HAVE_RPVST_PLUS */

#ifdef HAVE_HA
HA_CDR_REF mstp_instance_port_cdr_ref; /* CDR ref for mstp_instance_port */
```

```
HA_CDR_REF errdisable_timer_cdr_ref; /* CDR ref for errdisable timer */
  HA_CDR_REF port_timer_cdr_ref; /* CDR ref for port_timer */
  HA_CDR_REF edge_delay_timer_cdr_ref; /* CDR ref for edge_delay timer */
HA_CDR_REF forward_delay_timer_cdr_ref; /* CDR ref for forward_delay timer
*/
HA_CDR_REF hello_timer_cdr_ref; /* CDR ref for hello timer */
  HA_CDR_REF migrate_timer_cdr_ref; /* CDR ref for migrate timer */
HA_CDR_REF recent_backup_timer_cdr_ref; /* CDR ref for recent_backup timer
*/
HA_CDR_REF message_age_timer_cdr_ref; /* CDR ref for message_age timer */
HA_CDR_REF recent_root_timer_cdr_ref; /* CDR ref for recent_root timer */
HA_CDR_REF tc_timer_cdr_ref; /* CDR ref for tc timer */
HA_CDR_REF hold_timer_cdr_ref; /* CDR ref for hold timer */
#endif /* HAVE_HA */
/* statistics variables */
u_int32_t conf_bpdu_sent;
u_int32_t conf_bpdu_rcvd;
u_int32_t tcn_bpdu_sent;
u_int32_t tcn_bpdu_rcvd;
u_int32_t src_mac_count;
u_int32_t similar_bpdu_cnt;
u_int32_t msg_age_timer_cnt;
 u_int32_t total_src_mac_count;

  /* BPDU Counters */
  ut_int64_t bpdu_discards_rx;
  ut_int64_t bpdu_discards_tx;
  ut_int64_t bpdu_guard_events;
  ut_int64_t enableBPDUrx_discards_count;
  ut_int64_t enableBPDUtx_discards_count;
};
```

Application Programming Interface

The functions defined in this subsection are used to send messages to and receive messages from NSM, called by the MSPT CLI commands, or used to manage MSTP MIB.

API Function	Description
mstp_nsm_recv_bridge_add	calls the bridge add message from NSM.
mstp_nsm_recv_bridge_add_if	Adds a port to the bridge by calling the appropriate command.
mstp_nsm_recv_bridge_add_vlan	Processes a VLAN add event.
mstp_nsm_recv_bridge_ageing_time_set	Calls the message from NSM to update the bridge ageing time.
mstp_nsm_recv_bridge_delete	calls the bridge delete message from NSM.
mstp_nsm_recv_bridge_delete_if	Deletes a port from the bridge by calling the appropriate command.

API Function	Description
mstp_nsm_recv_bridge_delete_vlan	Processes the VLAN delete event.
mstp_nsm_recv_bridge_if_state_sync_req	Synchronizes the STP port states with NSM.
mstp_nsm_recv_interface_delete	Calls the interface name to delete messages from NSM.
mstp_nsm_recv_interface_state	Retrieves the state of an interface.
mstp_nsm_recv_interface_update	Calls the interface name and updates the message from NSM.
mstp_nsm_recv_svlan_add_ce_port	Triggers when the addition of a CVLAN registration entry results in adding a mapping for a new SVLAN.
mstp_nsm_recv_svlan_delete_ce_port	Triggered when the deletion of a CVLAN registration entry results in deletion of all mappings for an SVLAN.
mstp_nsm_recv_vlan_add_port	Processes the port being added to a VLAN event.
mstp_nsm_recv_vlan_delete_port	Processes the port being deleted from a VLAN event.
mstp_nsm_recv_vlan_port_type	Processes the change of port type message from NSM.
mstp_nsm_send_ageing_time	Calls the NSM client send API to send message to NSM.
mstp_nsm_send_port_state	Calls the NSM client send API to send a message to NSM.
mstp_api_add_instance	Adds a bridge instance.
mstp_api_add_port	Adds a port to a bridge instance.
mstp_api_delete_port	Adds a port to a bridge instance.
mstp_api_delete_instance	Deletes a bridge instance
mstp_api_delete_instance_vlan	Deletes a bridge instance associated to a VLAN
mstp_api_disable_bridge	Disables a bridge and deactivates the spanning tree protocol
mstp_api_enable_bridge	Enables a bridge
mstp_api_get_enable_bpdurx	Gets the setting that specifies whether BPDU receipt is enabled on a specified port
mstp_api_get_enable_bpdutx	Gets the setting that specifies whether BPDU transmission is enabled on a specified port
mstp_api_get_loopguard_effective_status	Gets the loopguard effective status of a port
mstp_api_get_loopguard_status	Gets the loopguard administrative state of a port
mstp_api_get_msti_port_path_cost	Gets the path cost for a port for an MSTP Instance (MSTI)
mstp_api_get_msti_port_path_cost_method	Gets the type of path cost recalculation (automatic or manual) for an MSTI

API Function	Description
mstp_api_get_port_errdisable_timeout_interval	Gets the BPDU guard timeout interval
mstp_api_get_port_errdisable_timeout_status	Gets the setting that specifies whether BPDU guard timeout control is enabled
mstp_api_get_port_forceversion	Gets the force version for a port
mstp_api_get_port_path_cost	Gets the path cost for a port
mstp_api_get_port_path_cost_method	Gets the path cost method for a port
mstp_api_get_port_role	Gets the role of a port
mstp_api_mcheck	Clears all detected protocols from the specified bridge
mstp_api_region_name	Sets the bridge region and recomputes related port roles
mstp_api_revision_number	Sets the configuration revision level for a bridge and recomputes related port roles
mstp_api_set_ageing_time	Sets the bridge dynamic aging time
mstp_api_set_auto_edge	Enables or disables the auto-edge feature for a port
mstp_api_set_bridge_errdisable_timeout_enable	Enables or disables the err-disable-timeout feature
mstp_api_set_bridge_errdisable_timeout_interval	Enables or disables the err-disable-interval feature
mstp_api_set_bridge_forceversion	Sets the force version for a bridge
mstp_api_set_bridge_portfast_bpduguard	Enables the portfast BPDU guard on a bridge and the oper_bpduguard of its ports
mstp_api_set_bridge_portfast_bpdufilter	Sets portfast BPDU filtering on a specified bridge and all ports that have the default setting
mstp_api_set_bridge_priority	Sets the bridge priority and updates the priority of ports that use this bridge as the designated bridge
mstp_api_set_msti_instance_restricted_role	Sets the restricted role for an MSTI
mstp_api_set_msti_instance_restricted_tcn	Sets the restricted TCN for an MSTI
mstp_api_set_msti_port_auto_path_cost	Sets the path cost recalculation to automatic mode for an MSTI
mstp_api_set_msti_port_path_cost	Sets the path cost for a port for an MSTI
mstp_api_set_msti_port_priority	Sets the priority of a bridge-group for an MSTI
mstp_api_set_pathcost_method	Sets the base for path cost
mstp_api_set_port_auto_path_cost	Sets the path cost recalculation to automatic mode for CIST
mstp_api_set_port_bpdufilter	Sets the port bpduguard filter for a CIST port

API Function	Description
mstp_api_set_port_bpduguard	Sets the port bpduguard for a CIST port
mstp_api_set_port_edge	Sets the port as an edge port connected to an end station
mstp_api_set_port_errdisable_timeout_interval	Sets the BPDU guard timeout interval
mstp_api_set_port_errdisable_timeout_status	Enables BPDU guard timeout control on an edge port
mstp_api_set_port_forceversion	Sets the force version for a port
mstp_api_set_port_hello_time	Sets the port hello time
mstp_api_set_port_p2p	Sets the link type of the port
mstp_api_set_port_path_cost	Sets the CIST path cost for a port
mstp_api_set_port_priority	Sets the priority of a port
mstp_api_set_port_restricted_role	Sets the restricted role for a port
mstp_api_set_port_restricted_tcn	Sets restricted TCN for a port
mstp_api_set_port_rootguard	Sets the root guard for a port
mstp_api_set_transmit_hold_count	Sets the transmit hold-count for a bridge
mstp_snmp_dot1dBridgeScalars	Returns the value of an SNMP variable in the dot1Bridge table
mstp_snmp_dot1dStpExtPortTable	Returns the value of an SNMP variable in dot1dStpExtPortTable
mstp_snmp_dot1dStpPortTable	Returns the value of an SNMP variable in dot1dStpPortTable
mstp_snmp_write_dot1dBridgeScalars	Returns the value of an SNMP variable in the dot1dBridge table
mstp_snmp_write_dot1dStpExtPortTable	Returns the value of an SNMP variable in dot1dStpExtPortTable
mstp_snmp_write_dot1dStpPortTable	Returns the value of an SNMP variable in dot1dStpPortTable
xstp_snmp_dot1dBasePortTable	Returns the value of an SNMP variable in dot1dBasePortTable
xstp_snmp_dot1dTpFdbTable	Returns the value of an SNMP variable in dot1dTpFdbTable
xstp_snmp_dot1dTpPortTable	Returns the value of an SNMP variable in dot1dTpPortTable.

mstp_nsm_recv_bridge_add

This function receives the bridge add message from NSM.

Syntax

```
static int
mstp_nsm_recv_bridge_add (struct nsm_msg_header *header,
```

```
                                    void *arg, void *message)
```

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

Zero (0) when function succeeds.

RESULT_ERROR when function fails.

mstp_nsm_recv_bridge_add_if

This function adds one or more ports to the bridge by calling the appropriate `bridge_add_port` command API.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

Zero (0) always

mstp_nsm_recv_bridge_add_vlan

This function processes the VLAN Add event.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.

`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_bridge_ageing_time_set

This function receives the message from NSM to update the bridge aging time.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when the bridge for which a message was received is not found.

mstp_nsm_recv_bridge_delete

This function receives the bridge delete message from NSM.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

Zero (0) always

mstp_nsm_recv_bridge_delete_if

This function deletes one or more ports from the bridge by calling the appropriate `bridge_delete_port` command API.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

Zero (0) always

mstp_nsm_recv_bridge_delete_vlan

This function processes the VLAN delete event.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_bridge_if_state_sync_req

This function synchronizes the STP port states with NSM.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_interface_delete

This function receives the interface name to delete messages from NSM.

Syntax

```
static int
mstp_nsm_recv_interface_delete (struct interface *ifp)
```

Input Parameters

`ifp`	Reference to interface structure.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_interface_state

This function retrieves the state of an interface, based on whether the link message received is `NSM_MSG_LINK_UP` or `NSM_MSG_LINK_DOWN`, respectively.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when the port for which a message was received is not found.

mstp_nsm_recv_interface_update

This function receives the interface name and updates the message from NSM.

Syntax

```
static int
mstp_nsm_recv_interface_update (struct interface *ifp)
```

Input Parameters

`ifp`	Reference to interface structure.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_svlan_add_ce_port

This function triggers when the addition of a CVLAN registration entry results in adding a mapping for a new SVLAN. This results in creating a logical provider edge port, and adding it to the spanning tree domain of the customer edge port.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_svlan_delete_ce_port

This function is triggered when the deletion of a CVLAN registration entry results in deletion of all mappings for an SVLAN. This results in deleting a logical provider edge port, and deleting it from the spanning tree domain of the customer edge port.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_vlan_add_port

This function processes the port being added to a VLAN event.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_recv_vlan_delete_port

This function processes the port being deleted from a VLAN event.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Return Values

RESULT_OK always

mstp_nsm_recv_vlan_port_type

This function processes the change of port type message from NSM. If it is a customer edge port, it creates a spanning tree domain for the customer edge port.

Syntax

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

RESULT_OK always

mstp_nsm_send_ageing_time

This function calls the NSM client send API `(nsm_client_send_bridge_msg)` to send a message to NSM.

Syntax

```
int
mstp_nsm_send_ageing_time (const char *const br_name, s_int32_t ageing_time);
```

Input Parameters

`br_name`	Name of the bridge.
`ageing_time`	The new value of aging time for the bridge

Output Parameters

None

Return Values

Zero (0) if function succeeds

Negative value when function fails.

mstp_nsm_send_port_state

This function calls NSM client send API `(nsm_client_send_stp_message)` to send a message to NSM.

```
int
mstp_nsm_send_port_state (u_int32_t ifindex, int port_state);
```

Input Parameters

`ifindex`	The index of the interface on which the data is to be sent
`port_state`	The state of the port

Output Parameters

None

Return Values

Zero (0) if function succeeds

Negative value when function fails.

mstp_sock_init

This function calls the PAL API to create a socket of type `AF_STP`. If successful, it returns the socket descriptor to the protocol.

Syntax

```
pal_sock_handle_t mstp_sock_init (struct lib_globals *zg);
```

Input Parameters

`zg`	Name of the bridge.

Output Parameters

None

Return Values

Pocket file descriptor value is returned in the opening a socket when the function succeeds

Negative value error number when function fails.

mstp_recv

This function calls the PAL socket API to receive data from the socket previously opened by socket initialization. It also calls the BPDU handler to handle the packet received from the socket.

Syntax

```
int
mstp_recv (struct thread *thread);
```

Input Parameters

thread	The pointer to the thread context which called the receive

Output Parameters

None

Return Values

-1 if the socket descriptor is not initialized

Negative value of error number if the socket receive returns an error

Otherwise, the function returns the number of bytes read from the socket

mstp_send

This function calls the PAL socket API to send BPDU.

```
int
mstp_send (const char *const brname, u_char *src_addr, const char *dest_addr,
           const u_int16_t vid, const u_int32_t ifindex, unsigned char *data,
           int length);
```

Input Parameters

brname	Name of the bridge.
src_addr	Source MAC address to be used to send.
dest_addr	Destination MAC address to which the data is to be sent.
ifindex	Index of interface on which the data is to be sent.
data	Data buffer pointer.
length	Length of the data pointed to by data.

Output Parameters

None

Return Values

RESULT_ERROR if the socket is not initialized, data is NULL, or the function failed to send data

Otherwise, the function returns the number of bytes sent over the socket

mstp_api_add_instance

This function adds a bridge instance.

Syntax

```
int
mstp_api_add_instance (char * name, int instance, mstp_vid_t vid,
                       u_int32_t vlan_range_indx)
```

Input Parameters

`name` Name of the bridge.

`instance` Instance identifier <1-63> or one of these constants from `mstpd/mstp_config.h`:

`SPBM_INSTANCE_ID`
Shortest Path Bridging - MAC

`MSTP_TE_MSTID`
Traffic Engineering MSTI.

`vid` VLAN identifier.

`vlan_range_indx`
VLAN range index.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_INSTANCE_IN_USE_ERR when MSTP instance is in use.

MSTP_ERR_BRIDGE_NOT_FOUND when the bridge cannot be found

mstp_api_add_port

This function adds a port to a bridge instance.

Syntax

```
int
mstp_api_add_port (char * name, char *ifname, u_int16_t svid,
                   int instance, u_int8_t spanning_tree_disable)
```

Input Parameters

`name` Name of the bridge.

`ifname` The interface name.

`svid` Service VLAN ID.

`instance` Instance identifier <1-63> or one of these constants from `mstpd/mstp_config.h`:

`SPBM_INSTANCE_ID`
Shortest Path Bridging - MAC

`MSTP_TE_MSTID`
Traffic Engineering MSTI.

`spanning_tree_disable`

Whether spanning-tree is enabled or disabled on the interface.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_INSTANCE_ALREADY_BOUND when the interface is already bound to the instance

mstp_api_delete_port

This function deletes a port from the bridge instance.

Syntax

```
int
mstp_api_delete_port (char * name, char *ifname, int instance, u_int16_t svid,
                      int force, bool_t notify_fwd)
```

Input Parameters

`name` Name of the bridge.

`ifname` The interface name.

`instance` Instance identifier <1-63> or one of these constants from `mstpd/mstp_config.h`:

`SPBM_INSTANCE_ID`

Shortest Path Bridging - MAC

`MSTP_TE_MSTID`

Traffic Engineering MSTI.

`svid` Service VLAN ID.

`force` Whether to delete the port even if another instance references it.

`notify_fwd` Reserved for future use.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when the bridge cannot be found

MSTP_ERR_PORT_NOT_FOUND when the interface cannot be found

mstp_api_delete_instance

This function deletes a bridge instance.

Syntax

```
int
mstp_api_delete_instance (char * name, int instance)
```

Input Parameters

`name` Name of the bridge.

`instance` Instance identifier <1-63> or one of these constants from `mstpd/mstp_config.h`:

`SPBM_INSTANCE_ID`
Shortest Path Bridging - MAC

`MSTP_TE_MSTID`
Traffic Engineering MSTI.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when the bridge cannot be found

MSTP_ERR_INSTANCE_NOT_FOUND when the instance cannot be found

mstp_api_delete_instance_vlan

This function deletes a bridge instance associated with a VLAN.

Syntax

```
int
mstp_api_delete_instance_vlan (char *name, int instance, mstp_vid_t vid)
```

Input Parameters

`name` Name of the bridge.

`instance` Instance identifier <1-63> or one of these constants from `mstpd/mstp_config.h`:

`SPBM_INSTANCE_ID`
Shortest Path Bridging - MAC

`MSTP_TE_MSTID`
Traffic Engineering MSTI.

`vid` VLAN identifier.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when the bridge cannot be found

MSTP_ERR_BR_INST_ISID_MAPPED_TO_BVLAN when the VLAN is associated with an I-SID

mstp_api_disable_bridge

This function disables a bridge and deactivates the spanning tree protocol.

Syntax

```
int
mstp_api_disable_bridge (char * name, u_int8_t br_type, bool_t bridge_forward)
```

Input Parameters

`name`	Name of the bridge.
`br_type`	Type of bridge (STP, RSTP, MSTP).
`bridge_forward`	Whether the CIST state of all ports attached to the bridge is STATE_FORWARDING (1 = PAL_TRUE) or not (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when bridge is disabled

mstp_api_enable_bridge

This function enables a bridge.

Syntax

```
int
mstp_api_enable_bridge (char * name, u_int8_t br_type)
```

Input Parameters

`name`	Name of the bridge.
`br_type`	Type of bridge as defined in mstpd/mstp_bridge.h:

`NSM_BRIDGE_TYPE_STP`

`NSM_BRIDGE_TYPE_STP_VLANAWARE`

`NSM_BRIDGE_TYPE_RSTP`

`NSM_BRIDGE_TYPE_RSTP_VLANAWARE`

`NSM_BRIDGE_TYPE_MSTP`

`NSM_BRIDGE_TYPE_PROVIDER_RSTP`

`NSM_BRIDGE_TYPE_PROVIDER_MSTP`

`NSM_BRIDGE_TYPE_CE`

`NSM_BRIDGE_TYPE_RPVST_PLUS`

`NSM_BRIDGE_TYPE_BACKBONE_RSTP`

`NSM_BRIDGE_TYPE_BACKBONE_MSTP`

`SMI_DONT_CHK_TYPE`

(Defined in lib/smi/client/smi_message.h)

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when bridge is enabled

mstp_api_get_enable_bpdurx

This function gets the setting that specifies whether BPDU receipt is enabled on a specified port.

Syntax

```
result_t
mstp_api_get_enable_bpdurx (char * br_name, char * if_name, bool_t *enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.

Output Parameters

`enabled`	Whether BPDU receipt is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the `br_name` does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_get_enable_bpdutx

This function gets the setting that specifies whether BPDU transmission is enabled on a specified port.

Syntax

```
result_t
mstp_api_get_enable_bpdutx(char * br_name, char * if_name, bool_t *enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.

Output Parameters

`enabled` Whether BPDU transmission is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the name does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_get_loopguard_effective_status

This function gets the loopguard effective status of a port.

Syntax

```
result_t
mstp_api_get_loopguard_effective_status (u_int8_t *br_name,
                                         u_int8_t *if_name,
                                         u_int8_t *eff_status)
```

Input parameters

`br_name` Bridge name.

`if_name` Interface name.

Output parameters

`eff_status` The effective status of the port defined in mstpd/mstp_types.h:

`MSTP_PORT_LOOPGUARD_INACTIVE`

`MSTP_PORT_LOOPGUARD_ACTIVE`

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when an interface is not found.

mstp_api_get_loopguard_status

This function gets the loopguard administrative state of a port.

Syntax

```
result_t
mstp_api_get_loopguard_status (u_int8_t *br_name,
                               u_int8_t *if_name,
```

```
u_int8_t *status)
```

Input parameters

`br_name`	Bridge name.
`if_name`	Interface name.

Output parameters

`status`	The status of the port defined in mstpd/mstp_types.h:

`MSTP_PORT_LOOPGUARD_DISABLED`

`MSTP_PORT_LOOPGUARD_ENABLED`

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

mstp_api_get_msti_port_path_cost

This function gets the path cost for a port for an MSTI.

Syntax

```
int
mstp_api_get_msti_port_path_cost (char *name, char *ifName, int instance,
u_int32_t *cost)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	Interface name.
`instance`	MSTP Instance.

Output Parameters

`cost`	Path cost as recommended in IEEE 802.1D 2004, which is in the range of 1–200,000,000, with lower values used for higher speed links.

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_INSTANCE_NOT_FOUND if the MSTI instance does not exist.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

mstp_api_get_msti_port_path_cost_method

This function gets the type of path cost recalculation (automatic or manual) for an MSTI.

Syntax

```
int
mstp_api_get_msti_port_path_cost_method (char *name, char *ifName,
int instance, u_int8_t *pathcost_method)
```

Input parameters

`name`	Bridge name.
`ifName`	Interface name.
`instance`	MSTP Instance.

Output parameters

`pathcost_method`

Type of path cost recalculation:

`1`	Automatic
`0`	Manual

Return values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when the specified bridge is not found.

MSTP_ERR_INSTANCE_NOT_FOUND when the specified MSTI instance is not found.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

mstp_api_get_port_errdisable_timeout_interval

This function gets the BPDU guard timeout interval.

Syntax

```
result_t
mstp_api_get_port_errdisable_timeout_interval (u_int8_t *br_name,
                                               u_int8_t *if_name,
                                               u_int32_t *timeout)
```

Input parameters

`br_name`	Bridge name.
`if_name`	Interface name.

Output parameters

`timeout`	The BPDU guard timeout interval in seconds.

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when a specified interface is not found.

mstp_api_get_port_errdisable_timeout_status

This function gets the setting that specifies whether BPDU guard timeout control is enabled.

Syntax

```
result_t
mstp_api_get_port_errdisable_timeout_status (u_int8_t *br_name,
                                             u_int8_t *if_name,
                                             u_int8_t *status)
```

Input parameters

`br_name` Bridge name.

`if_name` Interface name.

Output parameters

`status` The status of the BPDU guards timeout control:

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_DISABLED`

Timeout is disabled on a port.

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_ENABLED`

Timeout is enabled on a port.

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_DEFAULT`

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when a specified interface is not found.

mstp_api_get_port_forceversion

This function gets the force version for a port.

Syntax

```
int
mstp_api_get_port_forceversion (char *name, char *ifName, s_int32_t *version)
```

Input Parameters

`name` Name of the bridge.

`ifName` The interface name.

Output Parameters

`version` The version identifier; one of these constants from mstpd/mstp_config.h:

`BR_VERSION_STP`
STP

`BR_VERSION_RSTP`
RSTP

`BR_VERSION_MSTP`
MSTP

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_get_port_path_cost

This function gets the path cost for a port.

Syntax

```
int
mstp_api_get_port_path_cost (char *name, char *ifName, u_int16_t svid,
u_int32_t *cost)
```

Input Parameters

`name` Name of the bridge.

`ifName` Interface name.

`svid` SVLAN identifier.

Output Parameters

`cost` Path cost as recommended in IEEE 802.1D 2004, which is in the range of 1–200,000,000, with lower values used for higher speed links.

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

mstp_api_get_port_path_cost_method

This function gets the type of path cost recalculation (automatic or manual).

Syntax

```
int
mstp_api_get_port_path_cost_method (char *name, char *ifName, u_int16_t svid,
u_int8_t *pathcost_method)
```

Input parameters

`name`	Bridge name.
`ifName`	Interface name.
`svid`	SVLAN identifier.

Output parameters

`pathcost_method`

Type of path cost recalculation:

`1`	Automatic
`0`	Manual

Return values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when the specified bridge is not found.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

mstp_api_get_port_role

This function gets the role of a port.

Syntax

```
result_t
mstp_api_get_port_role(u_int8_t *br_name, u_int8_t *if_name,
enum port_role *role)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	The interface name.

Output Parameters

`role` The role; one of these constants from the enum port_role in mstpd/mstp_types.h:

- `ROLE_MASTERPORT`
- `ROLE_ALTERNATE`
- `ROLE_ROOTPORT`
- `ROLE_DESIGNATED`
- `ROLE_DISABLED`
- `ROLE_BACKUP`

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_BRIDGE_NOT_FOUND when specified bridge is not found.

MSTP_ERROR_GENERAL when parameters are wrongly passed.

MSTP_ERR_INTERFACE_NOT_FOUND when specified interface is not passed.

MSTP_ERR_PORT_NOT_FOUND when specified port is not passed.

mstp_api_mcheck

This function clears all detected protocols from the specified bridge.

Syntax

```
int
mstp_api_mcheck (char *name, char *ifName)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_region_name

This function sets the bridge region and recomputes related port roles.

Syntax

```
int
mstp_api_region_name (char *name, char *region_name)
```

Input Parameters

`name`	Name of the bridge.
`region_name`	The name of the region.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_revision_number

This function sets the configuration revision level for a bridge and recomputes related port roles.

Syntax

```
int
mstp_api_revision_number (char * name, u_int16_t rev_num)
```

Input Parameters

`name`	Name of the bridge.
`rev_num`	The MSTP configuration revision level used for this bridge.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_ageing_time

This function sets the bridge dynamic aging time.

Syntax

```
int
mstp_api_set_ageing_time (char * name, s_int32_t ageing_time)
```

Input Parameters

`name`	Name of the bridge.
`ageing_time`	The aging time in seconds.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_auto_edge

This function enables or disables the auto-edge feature for a port.

Syntax

```
int
mstp_api_set_auto_edge (char * name, char * ifName,
                        int to_be_enabled)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.
`to_be_enabled`	Whether the auto edge feature is enabled or disabled.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface,

RESULT_ERROR when function fails.

mstp_api_set_bridge_errdisable_timeout_enable

This function enables or disables the errdisable-timeout feature.

Syntax

```
int
mstp_api_set_bridge_errdisable_timeout_enable (char *br_name,
                                               bool_t enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`enabled`	Whether the feature is enabled.

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when function succeeds.

mstp_api_set_bridge_errdisable_timeout_interval

This function sets the errdisable-timeout interval on the bridge and all ports belonging to the bridge where the port's errdisable_timeout_enable is set to MSTP_PORT_BPDUGUARD_TIMEOUT_DEFAULT.

Syntax

```
int
mstp_api_set_bridge_errdisable_timeout_interval (char *br_name,
                                                 s_int32_t timeout)
```

Input Parameters

`br_name`	Name of the bridge.
`timeout`	The error disable timeout interval in seconds.

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when function succeeds.

mstp_api_set_bridge_forceversion

This function sets the force version for a bridge.

Syntax

```
int
mstp_api_set_bridge_forceversion (char *name, s_int32_t version)
```

Input Parameters

`name` Name of the bridge.

`version` The version identifier; one of these constants from `mstpd/mstp_config.h`:

`BR_VERSION_STP`
STP

`BR_VERSION_RSTP`
RSTP

`BR_VERSION_MSTP`
MSTP

`BR_VERSION_SPB`
Shortest Path Bridging

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_bridge_portfast_bpduguard

This function enables the portfast BPDU guard on a bridge and the oper_bpduguard of its ports that have the admin_bpduguard set to MSTP_PORT_PORTFAST_BPDUGUARD_DEFAULT.

Syntax

```
int
mstp_api_set_bridge_portfast_bpduguard (char *br_name,
                                        bool_t bpduguard_enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`bpduguard_enabled`	Whether the BPDU guard is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when the update of `bpduguard` succeeds

mstp_api_set_bridge_portfast_bpdufilter

This sets portfast BPDU filtering on a specified bridge and all ports that have the default setting.

Syntax

```
int
mstp_api_set_bridge_portfast_bpdufilter (char *br_name, bool_t enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`enabled`	Whether BPDU guard is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist.

RESULT_OK when the update of the BPDU guard succeeds

mstp_api_set_bridge_priority

This function sets the bridge priority and updates the priority of ports that use this bridge as the designated bridge. The root bridge selection may change as a result of calling this function.

Syntax

```
int
mstp_api_set_bridge_priority (char * name, u_int32_t new_priority)
```

Input Parameters

`name`	Name of the bridge.
`new_priority`	Priority that must be evenly divisible by MSTP_BRIDGE_PRIORITY_MULTIPLIER (defined in mstpd/mstp_bridge.h) and not greater than MSTP_MAX_BRIDGE_PRIORITY (defined in mstpd/mstp_config.h).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_BRIDGE_NOT_FOUND when no bridge with the supplied name is found

MSTP_ERR_PRIORITY_VALUE_WRONG when `new_priority` is not evenly divisible by MSTP_BRIDGE_PRIORITY_MULTIPLIER

MSTP_ERR_PRIORITY_OUTOFBOUNDS when `new_priority` is greater than MSTP_MAX_BRIDGE_PRIORITY

mstp_api_set_enable_bpdurx

This function enables or disables BPDU receipt on a specified port.

Syntax

```
result_t
mstp_api_set_enable_bpdurx (char * br_name, char * if_name, bool_t enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.
`enabled`	Whether BPDU receipt is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the `br_name` does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_set_enable_bpdutx

This function enables or disables BPDU transmission on a specified port.

Syntax

```
result_t
mstp_api_set_enable_bpdutx(char * br_name, char * if_name, bool_t enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.

`enabled`	Whether BPDU transmission is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the name does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_set_forward_delay

This function sets the forward delay interval (in seconds) for a bridge.

Syntax

```
int
mstp_api_set_forward_delay (char * name, s_int32_t forward_delay)
```

Input Parameters

`name`	Name of the bridge.
`forward_delay`	The bridge's forward delay interval in seconds. The value must be in the range of MSTP_MIN_BRIDGE_FWD_DELAY to MSTP_MAX_BRIDGE_FWD_DELAY defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_hello_time

This function sets the bridge hello interval for both MSTP and RSTP bridges.

Syntax

```
int
mstp_api_set_hello_time (char * name, s_int32_t hello_time)
```

Input Parameters

`name`	Name of the bridge.
`hello_time`	The hello interval in seconds that must be in the range of MSTP_MIN_BRIDGE_HELLO_TIME to MSTP_MAX_BRIDGE_HELLO_TIME defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_HELLO_NOT_CONFIGURABLE when bridge type is RSTP or MSTP.

mstp_api_set_loopguard_status

This function sets the loopguard administrative state of a port.

Syntax

```
result_t
mstp_api_set_loopguard_status (u_int8_t *br_name,
                               u_int8_t *if_name,
                               u_int8_t status)
```

Input parameters

`br_name`	Bridge name.
`if_name`	Interface name.
`status`	The status of the port defined in mstpd/mstp_types.h:

`MSTP_PORT_LOOPGUARD_DISABLED`

`MSTP_PORT_LOOPGUARD_ENABLED`

Output parameters

None

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_MSTI_CONFIGURED_ON_PORT when an MSTI configured on the ported

mstp_api_set_max_age

This function sets the maximum age for a bridge.

Syntax

```
int
mstp_api_set_max_age (char * name, s_int32_t max_age)
```

Input Parameters

`name`	Name of the bridge.
`max_age`	The maximum age in the range of MSTP_MIN_BRIDGE_MAX_AGE to MSTP_MAX_BRIDGE_MAX_AGEs. The default is 20 seconds.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_max_hops

This function sets the maximum hops of the bridge.

Syntax

```
int
mstp_api_set_max_hops (char * name, s_int32_t max_hops)
```

Input Parameters

`name`	Name of the bridge.
`max_hops`	The maximum hops in the range MSTP_MIN_BRIDGE_MAX_HOPS to MSTP_MAX_BRIDGE_MAX_HOP defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist or if `max_hops` is not within range

RESULT_OK when `max_hops` is correctly set

mstp_api_set_msti_bridge_priority

This function sets the bridge priority and the priority of each port that uses this bridge as the designated bridge. The root bridge selection may change as a result of calling this function.

Syntax

```
int
mstp_api_set_msti_bridge_priority (char * name, int instance,
                                   u_int32_t new_priority)
```

Input Parameters

`name`	Name of the bridge.
`instance`	Instance identifier <1-63> or specify `SPBM_INSTANCE_ID` (defined in `mstpd/mstp_config.h`) for Shortest Path Bridging - MAC.

`new_priority`	Priority that must be evenly divisible by `MSTP_BRIDGE_PRIORITY_MULTIPLIER` and not greater than `MSTP_MAX_BRIDGE_PRIORITY` (both constants are defined in `mstpd/mstp_config.h`).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when the named bridge cannot be found

MSTP_ERR_NOT_SPB_BRIDGE when the bridge is not enabled for SPB.

MSTP_ERR_INSTANCE_OUTOFBOUNDS when the instance ID is less than MST_MIN_INSTANCE or greater than the greatest instance ID for the bridge

MSTP_ERR_INSTANCE_NOT_FOUND when the instance is not found.

MSTP_ERR_PRIORITY_VALUE_WRONG when wrong value is passed as priority.

MSTP_ERR_PRIORITY_OUTOFBOUNDS when priority value is out of bound.

mstp_api_set_msti_instance_restricted_role

This function sets the restricted role for an MSTI

Syntax

```
int
mstp_api_set_msti_instance_restricted_role (char *name, char *ifName,
                                            int instance,
                                            bool_t restricted_role)
```

Input Parameters

`name`	Bridge name.
`ifName`	Interface name.
`instance`	MSTP Instance.
`restricted_role`	Whether the instance has a restricted role (TRUE) or not (FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when function fails.

RESULT_OK when function succeeds.

mstp_api_set_msti_instance_restricted_tcn

This function sets the restricted Topology Change Notification (TCN) for an MSTI.

Syntax

```
int
mstp_api_set_msti_instance_restricted_tcn (char *name, char *ifName,
                                           int instance,
                                           bool_t restricted_tcn)
```

Input Parameters

`name`	Bridge name.
`ifName`	Interface name.
`instance`	MSTP Instance.
`restricted_tcn`	Whether the instance has a restricted TCN (TRUE) or not (FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when function fails.

RESULT_OK when function succeeds.

mstp_api_set_msti_port_auto_path_cost

This function sets the path cost recalculation to automatic mode for an MSTI.

Syntax

```
int
mstp_api_set_msti_port_auto_path_cost (char *name, char *ifName, int instance)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	Interface name.
`instance`	MSTP Instance.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_INSTANCE_NOT_FOUND when the MSTI instance does not exist.

MSTP_ERR_PORT_NOT_FOUND when port does not exist.

mstp_api_set_msti_port_path_cost

This function sets the path cost for a port for an MSTI.

Syntax

```
int
mstp_api_set_msti_port_path_cost (char *name, char *ifName,
                                  int instance,
                                  u_int32_t cost)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.
`instance`	MSTP Instance.
`cost`	Cost that must be between MSTP_MIN_PATH_COST and MSTP_MAX_PATH_COST defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_INTERFACE_NOT_FOUND when the interface is not found.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on interface.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge is not found.

MSTP_ERR_INSTANCE_NOT_FOUND when bridge instance is not found.

MSTP_ERR_PORT_NOT_FOUND when the interface is not found.

MSTP_ERR_PATH_COST_OUTOFBOUNDS if cost is not between MSTP_MIN_PATH_COST and MSTP_MAX_PATH_COST or MSTP_MAX_PATH_COST_SHORT

mstp_api_set_msti_port_priority

This function sets the priority of a bridge-group for an MSTI.

Syntax

```
int
mstp_api_set_msti_port_priority (char *name,
                                 char *ifName, int instance,
                                 s_int16_t priority)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.
`instance`	MSTP Instance.
`priority`	Priority that must be between MSTP_MIN_PORT_PRIORITY and MSTP_MAX_PORT_PRIORITY (both defined in mstpd/mstp_config.h) and must be evenly divisible by MSTP_BRIDGE_PORT_PRIORITY_MULTIPLIER (defined in mstpd/mstp_bridge.h).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge is not found.

MSTP_ERR_INSTANCE_NOT_FOUND when the instance is not found.

MSTP_ERR_PORT_NOT_FOUND when the interface is not found.

MSTP_ERR_PRIORITY_VALUE_WRONG when the priority is not evenly divisible by MSTP_BRIDGE_PORT_PRIORITY_MULTIPLIER.

MSTP_ERR_INSTANCE_OUTOFBOUNDS when the priority is outside the valid range.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

mstp_api_set_pathcost_method

The API sets the base for path cost. This function does not allow changing the path cost value base if there is a configured path cost for a port, since this value was likely set in accordance with a bandwidth-to-path cost matrix for the base configured for the bridge.

Syntax

```
int
mstp_api_set_pathcost_method (char *br_name, u_int8_t path_cost_method)
```

Input Parameters

`br_name` Name of the bridge.

`path_cost_method`

Base for the path cost; one of these constants from mstpd/mstp_api.h:

`MSTP_PATHCOST_SHORT`

Use the 16-bit default path cost from IEEE Std. 802.1D-1998.

`MSTP_PATHCOST_LONG`

Use the 32-bit default path cost from IEEE Std. 802.1t.

`MSTP_PATHCOST_DEFAULT`

Use 16-bit base for path cost for an STP bridge, 32-bit for all others.

Output Parameters

None

Return Values

MSTP_ERR when `path_cost_method` passed is not `short` or `long`

RESULT_OK when function succeeds.

MSTP_ERR_PATH_COST_BASE_OUTBOUNDS when the pathcost is out of bounds

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_PATHCOST_MANUAL_EXIST when there is a port with the path cost configured manually

mstp_api_set_port_auto_path_cost

This function sets the path cost recalculation to automatic mode for CIST.

Syntax

```
int
mstp_api_set_port_auto_path_cost (char *name, char *ifName, u_int16_t svid)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	Interface name.
`svid`	SVLAN identifier.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

mstp_api_set_port_bpdufilter

This function sets the port bpduguard filter for a CIST port.

Syntax

```
int mstp_api_set_port_bpdufilter (char * name,
                                  char * ifName,
                                  u_char portfast_bpdufilter)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	Interface name.
`portfast_bpdufilter`	BPDU filter for the port: MSTP_PORT_PORTFAST_BPDUFILTER_ENABLED MSTP_PORT_PORTFAST_BPDUFILTER_DISABLED MSTP_PORT_PORTFAST_BPDUFILTER_DEFAULT When the default is specified, the bridge BPDU filter is used.

Output Parameters

None

Return Values

RESULT_ERROR when a port is not found, when a port is not grouped to a bridge, or when a bridge does not exist.

RESULT_OK when port's BPDU filter is set for the CIST port

mstp_api_set_port_bpduguard

This function sets the port bpduguard for a CIST port.

Syntax

```
int
mstp_api_set_port_bpduguard (char * name, char * ifName,
                             u_char portfast_bpduguard)
```

Input Parameters

`name`	Name of the bridge.
`ifname`	Interface name.
`portfast_bpduguard`	Possible values including the following: MSTP_PORT_PORTFAST_BPDUGUARD_ENABLED MSTP_PORT_PORTFAST_BPDUGUARD_DISABLED MSTP_PORT_PORTFAST_BPDUGUARD_DEFAULT When the default is specified, the bridge BPDU guard is used.

Output Parameter

None

Return Values

RESULT_ERROR when a port is not found, when a port is not grouped to a bridge, or when a bridge does not exist.

RESULT_OK when port_fast bpdufilter is set for the CIST port

mstp_api_set_port_edge

This function sets the port as an edge port that is only connected to end stations.

Syntax

```
int
mstp_api_set_port_edge (char * name, char * ifName,
bool_t to_be_enabled)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The instance identifier.

to_be_enabled	Whether to enable (TRUE) or disable (FALSE) the port as an edge port.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on interface.

mstp_api_set_port_errdisable_timeout_interval

This function sets the BPDU guard timeout interval.

Syntax

```
result_t
mstp_api_set_port_errdisable_timeout_interval (u_int8_t *br_name,
                                               u_int8_t *if_name,
                                               u_int32_t timeout)
```

Input parameters

br_name	Bridge name.
if_name	Interface name.
timeout	The BPDU guard timeout interval in seconds.

Output parameters

None

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_TIMEOUT_OUTOFBOUNDS when there is an incorrect timeout interval.

MSTP_ERR_INTERFACE_NOT_FOUND when an interface is not found.

mstp_api_set_port_errdisable_timeout_status

This function enables or disables BPDU guard timeout control on an edge port.

Syntax

```
result_t
mstp_api_set_port_errdisable_timeout_status(u_int8_t *br_name,
```

```
                                                u_int8_t *if_name,
                                                u_int8_t status)
```

Input parameters

`br_name` Bridge name.

`if_name` Interface name.

`status` The status of the BPDU guard timeout control defined in mstpd/mstp_types:

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_DISABLED`

Timeout is disabled.

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_ENABLED`

Timeout is enabled.

`SMI_MSTP_PORTBPDUGUARD_TIMEOUT_DEFAULT`

Output parameters

None

Return values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

MSTP_ERR_PORT_NOT_FOUND when a specified port is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

MSTP_ERR_INTERFACE_NOT_FOUND when an interface is not found.

mstp_api_set_port_forceversion

This function sets the force version for a port.

Syntax

```
int
mstp_api_set_port_forceversion (char *name,
                                char *ifName,
                                s_int32_t version)
```

Input Parameters

`name` Name of the bridge.

`ifName` The interface name.

`version` The version identifier; one of these constants from mstpd/mstp_config.h:

`BR_VERSION_STP`

STP

`BR_VERSION_RSTP`

RSTP

`BR_VERSION_MSTP`

MSTP

Output Parameters

None

Return Values

RESULT_OK when bridge enable succeeds

RESULT_ERROR when bridge enable fails

mstp_api_set_port_hello_time

This function sets the port hello time.

Syntax

```
int
mstp_api_set_port_hello_time (char *name, char * ifName,
                              s_int32_t hello_time)
```

Input Parameters

`name`	Bridge name.
`ifName`	Interface name.
`hello_time`	The port hello time in seconds.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_HELLO_NOT_CONFIGURABLE when the interface is at the wrong level for the hello time setting

mstp_api_set_port_p2p

This function sets the link type of the port.

Syntax

```
int
mstp_api_set_port_p2p (char *name, char * ifName,
                       int is_p2p)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.
`is_p2p`	Whether the port is point-to-point. Possible values are 0 (unset), SMI_MSTP_PORT_P2P (enable P2P), or MSTP_ADMIN_LINK_TYPE_AUTO.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

mstp_api_set_port_path_cost

This function sets the CIST path cost for a port.

Syntax

```
int
mstp_api_set_port_path_cost (char *name, char *ifName,
                             u_int16_t svid, u_int32_t cost)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	The interface name.
`svid`	Service VLAN ID to be configured.
`cost`	Cost that must be between MSTP_MIN_PATH_COST and MSTP_MAX_PATH_COST defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_INTERFACE_NOT_FOUND when the interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge is not found.

MSTP_ERR_PORT_NOT_FOUND when the CIST port is not found.

MSTP_ERR_IF_PMIRROR_SET when port mirror is enabled on an interface.

MSTP_ERR_PATH_COST_OUTOFBOUNDS if cost is not between MSTP_MIN_PATH_COST and MSTP_MAX_PATH_COST or MSTP_MAX_PATH_COST_SHORT.

mstp_api_set_port_priority

This function sets the priority of a port.

Syntax

```
int
mstp_api_set_port_priority (char *name, char *ifName,
                            u_int16_t svid, s_int16_t priority)
```

Input Parameters

`name`	Name of the bridge
`ifName`	The interface name
`svid`	Service VLAN ID to be configured
`priority`	Priority that must be between MSTP_MIN_PORT_PRIORITY and MSTP_MAX_PORT_PRIORITY

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge is not found.

MSTP_ERR_PORT_NOT_FOUND when port is not found.

MSTP_ERR_PRIORITY_VALUE_WRONG when the priority is not evenly divisible by MSTP_BRIDGE_PORT_PRIORITY_MULTIPLIER

MSTP_ERR_PRIORITY_OUTOFBOUNDS when the priority is not between MSTP_MIN_PORT_PRIORITY and MSTP_MAX_PORT_PRIORITY

mstp_api_set_port_restricted_role

This function sets the restricted role for a port.

Syntax

```
int
mstp_api_set_port_restricted_role (char *name, char * ifName,
                                   bool_t restricted_role)
```

Input Parameters

`name`	Bridge name.
`ifName`	Interface name.
`restricted_role`	Whether the port's role is restricted (1 = PAL_TRUE) or not (0 =).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_port_restricted_tcn

This function sets restricted TCN for a port.

Syntax

```
int
mstp_api_set_port_restricted_tcn (char *name, char * ifName,
                                  bool_t restricted_tcn)
```

Input Parameters

`name`	Bridge name.
`ifName`	Interface name.
`restricted_tcn`	Whether the port operates with restricted TCN (1 = PAL_TRUE) or not (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

mstp_api_set_port_rootguard

This sets the root guard for a port.

Syntax

```
int
mstp_api_set_port_rootguard (char * name,
                                  char * ifName,
                                  bool_t enabled)
```

Input Parameters

`name`	Name of the bridge.
`ifName`	Interface name.
`enabled`	Whether root guard is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_ERROR when a port is not found, when a port is not grouped to a bridge, or when a bridge does not exist.

RESULT_OK when the port's root guard is set

mstp_api_set_transmit_hold_count

This function sets the transmit hold-count for a bridge.

Syntax

```
int
mstp_api_set_transmit_hold_count (char * name, unsigned char txholdcount)
```

Input Parameters

`name`	Name of the bridge.
`txholdcount`	Transmit hold count that must be between MSTP_MIN_BRIDGE_TX_HOLD_COUNT and MSTP_MAX_BRIDGE_TX_HOLD_COUNT defined in mstpd/mstp_config.h.

Output Parameters

None

Return Values

RESULT_ERROR when bridge does not exist or `txholdcount` is out of range

RESULT_OK when the update of `txholdcount` succeeds

Bridge MIB Support

The bridge and RSTP MIBs are implemented according to RFC 4188 (Managed Objects for Bridges). The sections below list the MIB attributes in the RFCs and show how they relate to functions in the ZebOS-XP API. Click a function name to jump to that reference topic.

RFC 4188

Supported objects:

Object Type	Syntax	Access	Functions
dot1dBaseBridgeAddress	MacAddress	read-only	
dot1dBaseNumPorts	INTEGER	read-only	
dot1dBasePortMtuExceededDiscards	Counter	read-only	
dot1dStpBridgeMaxAge	Timeout	read-write	mstp_api_set_max_age
dot1dStpBridgeHelloTime	Timeout	read-write	mstp_api_set_hello_time
dot1dStpBridgeForwardDelay	Timeout	read-write	mstp_api_set_forward_delay
dot1dStpPortForwardTransitions	Counter	read-only	
dot1dStpPortPathCost32	INTEGER	read-write	mstp_api_set_port_path_cost
dot1dStpPortState	INTEGER	read-only	
dot1dStpPortEnable	INTEGER	read-write	
dot1dStpPortDesignatedCost	INTEGER	read-only	
dot1dStpPortDesignatedBridge	BridgeId	read-only	

dot1dStpPortDesignatedPort	OCTET STRING	read-only	
dot1dStpPriority	INTEGER	read-write	mstp_api_set_bridge_priority
dot1dStpTimeSinceTopologyChange	TimeTicks	read-only	
dot1dStpTopChanges	Counter	read-only	
dot1dStpDesignatedRoot	BridgeId	read-only	
dot1dStpRootCost	INTEGER	read-only	
dot1dStpRootPort	INTEGER	read-only	
dot1dTpLearnedEntryDiscards	Counter	read-only	
dot1dTpAgingTime	INTEGER	read-write	mstp_api_set_ageing_time
dot1dTpPortInFrames	Counter	read-only	
dot1dTpPortOutFrames	Counter	read-only	
dot1dTpPortInDiscards	Counter	read-only	
dot1dTpFdbTable: dot1dTpFdbAddress dot1dTpFdbPort dot1dTpFdbStatus	 MacAddress Integer32 INTEGER	 read-only read-only read-only	
dot1dStpPortPriority	INTEGER	read-write	mstp_api_set_port_priority
dot1dStpPortDesignatedBridge	BridgeId	read-only	

MIB Functions

This section describes functions related to reading and writing MIBs.

mstp_snmp_dot1dBridgeScalars

This call returns the value of an SNMP variable in the dot1Bridge table.

Syntax.

```
unsigned char *
mstp_snmp_dot1dBridgeScalars (struct variable *vp, oid * name, size_t *
length,
int exact, size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).

`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

mstp_snmp_dot1dStpExtPortTable

This call returns the value of an SNMP variable in dot1dStpExtPortTable.

Syntax

```
unsigned char *
mstp_snmp_dot1dStpExtPortTable (struct variable *vp, oid * name, size_t *
length,
int exact, size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

mstp_snmp_dot1dStpPortTable

This call returns the value of an SNMP variable in dot1dStpPortTable.

Syntax

```
unsigned char *
mstp_snmp_dot1dStpPortTable (struct variable *vp, oid * name, size_t * length,
int exact, size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.

`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

mstp_snmp_write_dot1dBridgeScalars

This call returns the value of an SNMP variable in the dot1dBridge table.

Syntax

```
int
mstp_snmp_write_dot1dBridgeScalars (int action, u_char * var_val, u_char
var_val_type, size_t var_val_len, u_char * statP, oid * name, size_t length,
struct variable *v, u_int32_t vr_id)
```

Input Parameters

`action`	Unused.
`var_val`	Value of the variable.
`var_val_type`	Data type of the variable; this constant from lib/asn1.h, which corresponds to ASN.1's built-in simple type: ASN_INTEGER
`var_val_len`	Length of the variable.
`statP`	Unused.
`name`	Pointer to variable name (OID).
`length`	Length of the name.
`v`	Pointer to SNMP variable structure.
`vr_id`	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

mstp_snmp_write_dot1dStpExtPortTable

This call returns the value of an SNMP variable in dot1dStpExtPortTable.

Syntax

```
int
mstp_snmp_write_dot1dStpExtPortTable (int action, u_char * var_val,
u_char var_val_type, size_t var_val_len, u_char * statP, oid * name, size_t
length, struct variable *vp, u_int32_t vr_id)
```

Input Parameters

`action`	Unused.
`var_val`	Value of the variable.
`var_val_type`	Data type of the variable; this constant from lib/asn1.h, which corresponds to ASN.1's built-in simple type: ASN_INTEGER
`var_val_len`	Length of the variable.
`statP`	Unused.
`name`	Pointer to variable name (OID).
`length`	Length of the name.
`vp`	Pointer to SNMP variable structure.
`vr_id`	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

mstp_snmp_write_dot1dStpPortTable

This call returns the value of an SNMP variable in dot1dStpPortTable.

Syntax

```
int
mstp_snmp_write_dot1dStpPortTable (int action, u_char * var_val, u_char
var_val_type, size_t var_val_len, u_char * statP, oid * name, size_t length,
struct variable *vp,
u_int32_t vr_id)
```

Input Parameters

`action`	Unused.
`var_val`	Value of the variable.
`var_val_type`	Data type of the variable; this constant from lib/asn1.h, which corresponds to ASN.1's built-in simple type:
`ASN_INTEGER`	
`var_val_len`	Length of the variable.
`statP`	Unused.
`name`	Pointer to variable name (OID).
`length`	Length of the name.
`vp`	Pointer to SNMP variable structure.
`vr_id`	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

xstp_snmp_dot1dBasePortTable

This call returns the value of an SNMP variable in dot1dBasePortTable.

Syntax

```
unsigned char *
xstp_snmp_dot1dBasePortTable (struct variable *vp, oid * name, size_t *
length, int exact, size_t * var_len, WriteMethod ** write_method, u_int32_t
vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

xstp_snmp_dot1dTpFdbTable

This call returns the value of an SNMP variable in dot1dTpFdbTable.

Syntax

```
unsigned char *
xstp_snmp_dot1dTpFdbTable (struct variable *vp, oid * name, size_t * length,
int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

xstp_snmp_dot1dTpPortTable

This call returns the value of an SNMP variable in dot1dTpPortTable.

Syntax

```
unsigned char *
xstp_snmp_dot1dTpPortTable (struct variable *vp, oid * name, size_t * length,
int exact,
size_t * var_len, WriteMethod ** write_method,u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).

`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

CHAPTER 3 Rapid Per-VLAN Spanning Tree Plus

This chapter describes Rapid Per-VLAN Spanning Tree Plus (RPVST+) implementation in ZebOS-XP. This chapter includes an overview of RPVST+, a list of RPVST+ features, and a description of the supported RPVST+ functions.

Overview

The Rapid per VLAN Spanning Tree Plus (RPVST+) protocol builds an individual spanning-tree topology for each VLAN defined on a bridge. This topology runs on RPVST+ and 802.1q trunks. When a client puts a switch in RPVST+ mode, the switch can then support both RPVST+ and 802.1q inter-switch trunks. In RPVST+ mode, each VLAN runs its own independent spanning-tree instance. In addition, RPVST+ bridges can have different spanning-tree topologies for different VLANs within an autonomous switching domain. In ZebOS-XP, the RPVST+ module is implemented as part of the MSTP module.

Features

ZebOS-XP RPVST+ provides the following features:

- Concurrent support for RPVST+, STP, and MSTP with multiple bridge instances
- Support for default and native VLANs
- Easy scalability
- Load-balancing capability
- Rapid-convergence attributes

System Architecture

The following is a brief description of the RPVST+ system architecture.

Inter-switching Scenario

ZebOS-XP RPVST+ supports the following inter-switching scenario:

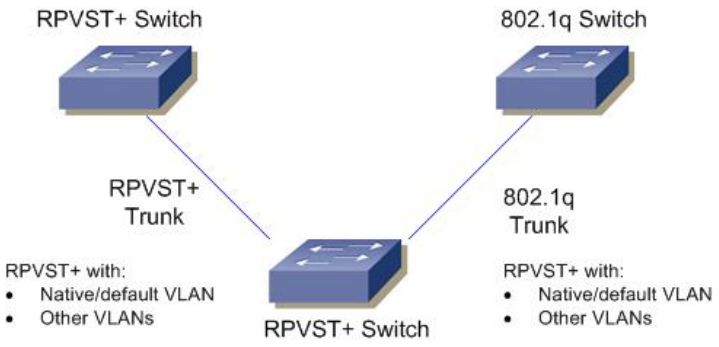

Figure 3-1: RPVST+ Inter-switching Scenario

Data Structures and Messaging

RPVST+ is an STP feature. Refer to Chapter 2, *Multiple Spanning Tree Protocol* for general information on system architecture, data structure information and system messaging for STP.

API

The API calls defined in this subsection are for the Rapid Per-VLAN Spanning Tree Plus (RPVST+) functionality in ZebOS-XP. It includes the following API calls:

Function	Description
rpvst_plus_api_add_port	Sets a port as a spanning-tree instance of a VLAN.
rpvst_plus_api_add_vlan	Sets a VLAN to be associated with a spanning tree instance.
rpvst_plus_api_set_msti_bridge_priority	Sets the priority on a spanning tree instance of a particular VLAN.
rpvst_plus_api_set_msti_port_auto_path_cost	Sets the path cost recalculation to automatic mode for an instance of MSTI related to RPVST.
rpvst_plus_api_set_msti_port_path_cost	Sets the priority level of a port on a VLAN.
rpvst_plus_api_set_msti_port_priority	Sets the priority level of a port on a VLAN.
rpvst_plus_api_set_msti_vlan_restricted_role	Sets restricted role on a VLAN.
rpvst_plus_api_set_msti_vlan_restricted_tcn	Sets restricted role on a VLAN.
rpvst_plus_api_vlan_delete	Unsets a VLAN that was associated with a spanning tree instance.
rpvst_plus_bridge_config_write	Performs an RPVST+ configuration write procedure.
rpvst_plus_config_write	Performs an RPVST+ configuration write procedure.
rpvst_plus_delete_port	Removes a port from a VLAN of a spanning tree.

rpvst_plus_api_add_port

This function sets a port as a spanning-tree instance of a VLAN.

Syntax

```
int
rpvst_plus_api_add_port(char *br_name, char *ifname, u_int16_t vid,
                        u_int8_t spanning_tree_disable)
```

Input Parameters

`br_name` Name of the bridge.

`ifname`	Interface name.
`vid`	VLAN ID to be configured.
`spanning_tree_disable`	Disables spanning tree on a port.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_add_vlan

This function sets a VLAN to be associated with a spanning tree instance. For each VLAN a spanning-tree instance is running when the VLAN is associated with a bridge when using the spanning-tree RPVST+ configuration mode.

Syntax

```
int
rpvst_plus_api_add_vlan (char *bridge_name, int vid)
```

Input Parameters

`bridge_name`	Name of the bridge.
`vid`	VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_set_msti_bridge_priority

This function sets the priority on a spanning tree instance of a particular VLAN. This function updates the bridge priority to the requested value. It also updates the priority of the ports that use this bridge as the designated bridge. The root bridge selection may change because of calling this function.

Syntax

```
int
rpvst_plus_api_set_msti_bridge_priority(char *bridge_name,
                                        int vid, u_int32_t priority)
```

Input Parameters

`bridge_name`	Name of the bridge.
`vid`	VLAN ID to be configured.

`priority`	The new priority value to be set on a VLAN.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_set_msti_port_auto_path_cost

This function sets the path cost recalculation to automatic mode for an instance of MSTI related to RPVST.

Syntax

```
int
rpvst_plus_api_set_msti_port_auto_path_cost (char *br_name,
                                             char *ifname, int vid)
```

Input Parameters

`br_name`	Name of the bridge.
`ifname`	The interface name.
`vid`	VLAN ID to be configured. Must be between 2 and 4094.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when the specified interface is not found.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge does not exist.

MSTP_ERR_INSTANCE_NOT_FOUND if the MSTI instance does not exist.

MSTP_ERR_PORT_NOT_FOUND when the specified port is not found.

MSTP_ERR_RPVST_VLAN_CONFIG_ERR when the VLAN does not exist.

rpvst_plus_api_set_msti_port_path_cost

This function sets the priority level of a port on a VLAN.

Syntax

```
int
rpvst_plus_api_set_msti_port_path_cost (char *br_name,
                                        char *ifname, int vid,
                                        u_int32_t path_cost)
```

Input Parameters

`br_name`	Name of the bridge.

ifname	The interface name.
vid	VLAN ID to be configured. Must be between 2 and 4094.
path_cost	Path cost. Must be between MSTP_MIN_PATH_COST and MSTP_MAX_PATH_COST.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_set_msti_port_priority

This function sets the priority level of a port on a VLAN.

Syntax

```
int
rpvst_plus_api_set_msti_port_priority (char *br_name,
                                       char *ifname, int vid,
                                       s_int16_t priority)
```

Input Parameters

br_name	Name of the bridge.
ifname	Interface name.
vid	RPVST+ VLAN to which this priority is applicable.
priority	Priority value. Must be between MSTP_MIN_PORT_PRIORITY and MSTP_MAX_PORT_PRIORITY.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_set_msti_vlan_restricted_role

This function sets restricted role on a VLAN.

Syntax

```
int
rpvst_plus_api_set_msti_vlan_restricted_role (char *br_name, char *ifname,
                                              int vid, bool_t flag)
```

Input Parameters

br_name	Bridge name.

`ifname`	Interface name.
`vid`	VLAN ID to be configured.
`flag`	Flag to be set.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_set_msti_vlan_restricted_tcn

This function sets restricted TCN on a VLAN.

Syntax

```
int
rpvst_plus_api_set_msti_vlan_restricted_tcn (char *br_name, char *ifname,
                                             int vid, bool_t flag)
```

Input Parameters

`br_name`	Bridge name.
`ifname`	Interface name.
`vid`	VLAN ID to be configured.
`flag`	Flag to be set.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_api_vlan_delete

This function unsets a VLAN that was associated with a spanning tree instance.

Syntax

```
int
rpvst_plus_api_vlan_delete (char *bridge_name, int vid)
```

Input Parameters

`bridge_name`	Name of the bridge.
`vid`	VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_bridge_config_write

This function performs an RPVST+ configuration write procedure.

Syntax

```
void
rpvst_plus_bridge_config_write (struct cli * cli,
                                struct mstp_bridge_instance *br_inst,
                                struct mstp_bridge *br)
```

Input Parameters

`cli`	CLI command.

Output Parameters

`br_instance`	Bridge instance
`br`	Bridge

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_config_write

This function performs an RPVST+ configuration write procedure.

Syntax

```
int
rpvst_plus_config_write (struct cli * cli)
```

Input Parameters

`cli`	CLI command.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

rpvst_plus_delete_port

This function removes a port from a VLAN of a spanning tree.

Syntax

```
int
rpvst_plus_delete_port(char *br_name, char *ifname, u_int16_t vid,
                       int force, bool_t notify_fwd)
```

Input Parameters

`name`	Name of the bridge.
`ifname`	Interface name.
`vid`	VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

CHAPTER 4 Flow Control

This chapter describes Flow Control in ZebOS-XP. This includes an overview of Flow Control, a list of Flow Control features, and a description of the supported Flow Control API calls.

Overview

All ZebOS-XP Spanning Tree modules support flow control (802.3x). Flow control is the process of managing the rate of data transmission between two nodes. Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process them. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

Features

The following are some of the features of Flow Control:

- Enhanced performance for Ethernet networks
- Lossless fabric for traffic on converged networks
- Virtual links by traffic class that includes pause-per-class technology, congestion management and event notification.
- Improved performance, including stability, enhanced throughput, and robustness

API

The following section lists and defines the Flow Control API calls. It includes the following API calls:

Function	Description
get_flow_control_statistics	Retrieves the 802.2x flow control statistics from a port.
port_add_flow_control	Sets the 802.3x flow control feature of a port.
port_delete_flow_control	Unsets the 802.3x flow control feature of an interface.

get_flow_control_statistics

This function retrieves the 802.2x flow control statistics from a port. This function uses the HAL API to get the flow control statistics.

Syntax

```
int
get_flow_control_statistics (struct interface *ifp,
                             enum smi_flow_control_dir *direction,
                             int *rxpause, int *txpause)
```

Input Parameters

`ifp`	Interface pointer.
`direction`	Indicates the direction of flow control. Possible values as defined in the lib/smi/client/smi_message.h file include the following: SMI_FLOW_CONTROL_OFF SMI_FLOW_CONTROL_SEND SMI_FLOW_CONTROL_RECEIVE SMI_FLOW_CONTROL_BOTH SMI_FLOW_CONTROL_UNCONFIGURE

Output Parameters

`rxpause`	Receive pause statistics.
`txpause`	Transmit pause statistics.

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

port_add_flow_control

This function sets the 802.3x flow control feature of a port.

Syntax

```
int
port_add_flow_control (struct interface *ifp, unsigned char direction)
```

Input Parameters

`ifp`	Interface pointer.
`direction`	Indicates the direction of flow control. Possible values as defined in the lib/smi/client/smi_message.h file include the following: SMI_FLOW_CONTROL_OFF SMI_FLOW_CONTROL_SEND SMI_FLOW_CONTROL_RECEIVE SMI_FLOW_CONTROL_BOTH

SMI_FLOW_CONTROL_UNCONFIGURE

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

port_delete_flow_control

This function unsets the 802.3x flow control feature of an interface.

Syntax

```
int
port_delete_flow_control (struct interface *ifp,
                          unsigned char direction)
```

Input Parameters

`ifp`	Interface pointer.
`direction`	Indicates the direction of flow control. Possible values as defined in the lib/smi/client/smi_message.h file include the following: SMI_FLOW_CONTROL_OFF SMI_FLOW_CONTROL_SEND SMI_FLOW_CONTROL_RECEIVE SMI_FLOW_CONTROL_BOTH SMI_FLOW_CONTROL_UNCONFIGURE

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

RESULT_NO_SUPPORT when function finds no support

CHAPTER 5 Layer 2 Gateway Port

This chapter describes the Layer 2 Gateway Port (L2GP) implementation in ZebOS-XP.

Overview

The L2GP solution helps separate different STP domains. In this solution, the user configures one or more ports as layer-two gateway ports, with each port defining the border of a domain in which the STP algorithm is active. The L2GP solution implements STP as a hello protocol and defines an L2GP as a regular port. A bridged network can be redundantly attached to a Provider Backbone Bridged Network (PBBN) by more than one layer two gateway ports. By communicating within their region and without influence from the PBBN, these ports connect to the PBBN through a single layer two gateway port, avoiding any bridging loops between the PBBN and the instance.

Figure 5-1 is an example of an L2GP configuration. In this configuration, ports 1 and 2 are configured with a bridge IDs of router 1 and router 2, respectively. Router 1 is better than router 2 and router 2 is better than any bridge in the A domain. As a result, port 1 is the forwarding (root) port, since its bridge continually receives the best BPDUs. Port 2 is then designated as the discarding port.

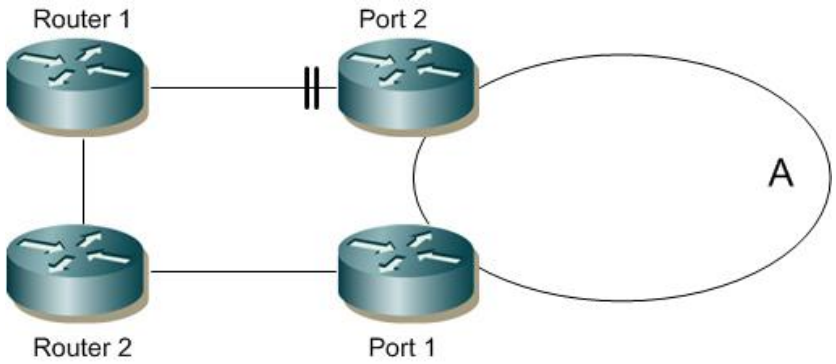

Figure 5-1: L2GP Configuration

pseudo-root Identity

A pseudo-root identity determines the open L2GP. The L2GP configured with the best pseudo-root identity is a root port in state forwarding. All others L2GPs are designated ports in state discarding. Thus, the port path cost value of L2GP is not important, since the open L2GP is determined by configuring pseudo-root identity.

Cost Control Switch

A non-zero path cost control switch called `l2gp_pathcost_control` is included in both the `mstp_bridge_info` and `mstp_bridge` structures. The configuration bit `MSTP_BRIDGE_CONFIG_L2GP_PATHCOST_CTRL` (defined in `mstpd/mstp_config.h`) is defined for these structures. When `l2gp_pathcost_control` is set to TRUE using the CLI or SMI API, the root path cost in the pseudo BPDU is set to the value of L2GP path cost. This is done in the `mstp_l2gp_prep_and_send_pseudoinfo` function.

API

Function	Description
mstp_api_get_bridge_l2gp_pathcost_control	Gets the setting of the L2GP non-zero path cost feature
mstp_api_set_bridge_l2gp_pathcost_control	Sets the L2GP non-zero path cost feature
mstp_api_get_isl2gp	Gets the status of an isL2gp object
mstp_api_set_isl2gp	Sets the status of an isL2gp object
mstp_api_get_l2gp_pseudorootid	Gets the pseudo-root identifier
mstp_api_set_l2gp_pseudorootid	Sets the pseudo-root identifier

mstp_api_get_bridge_l2gp_pathcost_control

This function gets the setting of the L2GP non-zero path cost feature (enabled or disabled).

Syntax

```
result_t
mstp_api_get_bridge_l2gp_pathcost_control (u_int8_t *br_name,
                                           bool_t *is_enabled)
```

Input Parameters

`br_name`	Name of the bridge.

Output Parameters

`is_enabled`	Whether this feature is enabled (either 1 = PAL_TRUE or 0 = PAL_FALSE).

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with `br_name` does not exist.

mstp_api_set_bridge_l2gp_pathcost_control

This function sets the L2GP non-zero path cost feature.

Syntax

```
result_t
mstp_api_set_bridge_l2gp_pathcost_control (u_int8_t *br_name,
                                           bool_t is_enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`is_enabled`	Whether non-zero path cost is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with `br_name` does not exist.

mstp_api_get_isl2gp

This function gets the status of an isL2gp object.

Syntax

```
result_t
mstp_api_get_isl2gp (u_int8_t *br_name, u_int8_t *if_name,
bool_t *is_enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.

Output Parameters

`is_enabled`	Whether the port is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE) as an L2GP port.

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the name does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_set_isl2gp

This function enables or disables the specified port as a Layer 2 Gateway Protocol port.

Syntax

```
result_t
mstp_api_set_isl2gp (u_int8_t *br_name, u_int8_t *if_name, bool_t is_enabled)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.
`is_enabled`	Whether the port is enabled (1 = PAL_TRUE) or disabled (0 = PAL_FALSE) as an L2GP port.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the name does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_get_l2gp_pseudorootid

This function gets the pseudo root identifier.

Note: A PseudoRootId is configured on a per port basis, only.

Syntax

```
result_t
mstp_api_get_l2gp_pseudorootid (u_int8_t *br_name, u_int8_t *if_name,
                                struct bridge_id *pseudo_rootid);
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.

Output Parameters

`pseudo_rootid`	The pseudo root identifier (of type bridge_id).

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the bridge_id does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

mstp_api_set_l2gp_pseudorootid

This function sets the bridge identifier (pseudo root ID) used by the L2 gateway protocol.

Note: A PseudoRootId is configured on a per port basis, only.

Syntax

```
result_t
mstp_api_set_l2gp_pseudorootid (u_int8_t *br_name, u_int8_t *if_name,
                                const struct bridge_id *pseudo_rootid)
```

Input Parameters

`br_name`	Name of the bridge.
`if_name`	Interface name.
`pseudo_rootid`	The bridge identifier used by the L2 gateway protocol.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

MSTP_ERR_INTERFACE_NOT_FOUND when interface does not exist.

MSTP_ERR_BRIDGE_NOT_FOUND when bridge with the name does not exist.

MSTP_ERR_PORT_NOT_FOUND when an MSTI port for the interface does not exist.

MSTP_ERROR_GENERAL when a bad is passed

CHAPTER 6 Port Mirroring

This chapter describes the port mirroring implementation in ZebOS-XP. This section includes an overview of port mirroring, a description of port mirroring, and a description of the supported API calls.

Overview

Port mirroring is used on network switches to send copies of all network packets seen on one switch port to a network monitoring connection on another switch port. This is commonly used for network appliances that require monitoring of network traffic, such as an intrusion-detection system. ZebOS-XP Spanning Tree protocol modules support port mirroring.

Data Structure

The following subsection list the port mirroring API data structure.

hal_port_mirror_direction

Port Mirroring direction enumeration. This data structure is defined in hal/hal_if.h.

Definition

```
enum hal_port_mirror_direction
  {
    HAL_PORT_MIRROR_DISABLE                =  0,
    HAL_PORT_MIRROR_DIRECTION_RECEIVE      = (1 << 0),
    HAL_PORT_MIRROR_DIRECTION_TRANSMIT     = (1 << 1),
    HAL_PORT_MIRROR_DIRECTION_BOTH         =
(HAL_PORT_MIRROR_DIRECTION_RECEIVE | HAL_PORT_MIRROR_DIRECTION_TRANSMIT)
```

Port Mirroring API

The section contains the functions for port mirroring.

Functions	Description
port_add_mirror_interface	Adds a port mirroring interface to the port mirror list
port_mirroring_list_dell	Deletes a port-mirroring list entry

port_add_mirror_interface

This function turns on port mirroring from the `from_ifindex` to the `to_ifindex` and specifies the direction of the packet flow that is to be mirrored.

Syntax

```
int
port_add_mirror_interface (struct interface *ifp_to, char *ifname,
                           int *mirror_direction)
```

Input Parameters

`ifp_to` Mirrored-to interface

`ifname` Mirrored-from interface name

`mirror_direction`

Direction to set for mirroring. Possible values as described in the hal_if.h file include the following:

`HAL_PORT_MIRROR_DISABLE`

0

`HAL_PORT_MIRROR_DIRECTION_RECEIVE`

1 << 0

`HAL_PORT_MIRROR_DIRECTION_TRANSMIT`

1 << 1

`HAL_PORT_MIRROR_DIRECTION_BOTH`

HAL_PORT_MIRROR_DIRECTION_RECEIVE | HAL_PORT_MIRROR_DIRECTION_TRANSMIT

Output Parameters

None

Return Values

RESULT_OK when function succeeds

RESULT_ERROR when function fails

RESULT_NO_SUPPORT when function finds no support

port_mirroring_list_del

This function removes a port from a mirroring list.

Syntax

```
int
port_mirroring_list_del(struct interface *to,
                        struct interface *from,
                        int *direction,
                        struct lib_globals *zg)
```

Input Parameters

`to` Mirrored-to interface

`from` Mirrored-from interface

`direction` Direction set for mirroring

zg	Library globals

Output Parameters

None

Return Values

RESULT_OK when function succeeds

RESULT_ERROR when function fails

RESULT_NO_SUPPORT when function finds no support

CHAPTER 7 Virtual LANs

This chapter describes Virtual LANs (VLANs) implementation in ZebOS-XP. This section includes an overview of VLANs, a list of the VLAN features, and a description of the supported external VLAN functions.

Overview

The VLAN modules offer a consistent network-wide management tools to manage virtual LANs (Local Area Networks) and bridged VLANs:

- VLAN bridging allows network devices to segment into VLANs, regardless of their physical location.
- VLANs, in accordance with IEEE 802.1Q, enable multiple bridged LANs to transparently share the same physical network link without leaking information between LANs. Traffic between VLANs is restricted to bridges that forward unicast, multicast, or broadcast traffic only on the LAN segments that serve the VLAN to which the traffic belongs.

The ZebOS-XP VLAN modules make it easy to administer logical groups of stations that can communicate as if they were on the same LAN. They make it easier to manage a move, add, delete, or other updates to members of these groups.

ZebOS-XP supports all IEEE 802.1D LAN MAC (Media Access Control) protocols, shared media, and point-to-point LANs. MAC (Media Access Control) Bridging allows multiple Local Area Networks (LANs) to be connected together. MAC bridging filters data sent between LAN segments, reduces network congestion, and allows networks to be partitioned for administrative purposes.

Identifying VLANS

VLAN identifiers (VIDs):

- Offer convenient, consistent, and network-wide methods for bridges to identify rules to classify user data for VLANs
- Extend source and destination MAC addresses by treating addressing for different VLANs independently
- Identify and select from multiple active topologies; and identify parameters that restrict access from one part of a network to another

When VLANs were originally defined in the 802.1Q, the number of unique VLAN identifiers was limited to 4096. In large provider networks, each subscriber needs a separate address, thus this limit could prevent a provider from having more than 4096 subscribers.

To overcome the 4096 VLAN identifier limit, the frame format for 802.1ad inserts an additional VLAN header into a single 802.1Q Ethernet frame. There are two types of VLAN headers:

- The C-VLAN or inner header which is closest to the payload portion of the frame identifies the customer VLAN
- The S-VLAN or outer header which is closest to the Ethernet header identifier the provider VLAN

The frame format for 802.1ad is also called “Q-in-Q” or “double tagged”.

With the two VLAN identifiers in combination for each provider-customer pair, it is possible to define up to 16,777,216 labels.

VLAN Prioritization (802.1p/Q)

ZebOS-XP includes priority signaling for traffic at the data-link layer. IEEE 802.1Q specifies a priority value of between 0 and 7 inclusive that can be used by QoS (Quality of Service) disciplines to differentiate traffic. Although this technique is often called “802.1p”, there is no standard by that name published by the IEEE. Instead, the technique is incorporated into 802.1Q standard which specifies the tag inserted into an Ethernet frame.

Port and Protocol Classification (802.1v)

Port and Protocol Classification is an amendment to 802.1Q to classify incoming packets by methods other than source port information, specifically, classification based on data-link-layer protocol identification.

VLAN Messaging

The following are the VLAN messages:

- NSM_MSG_SVLAN_ADD_CE_PORT when SVLAN added to CE port
- NSM_MSG_SVLAN_DELETE_CE_PORT when SLVAN is deleted from CE port
- NSM_MSG_VLAN_SET_PVID when VLAN is set on port
- NSM_MSG_VLAN_ADD_TO_PROTECTION Protection group add message from MSTP to NSM
- NSM_MSG_VLAN_DEL_FROM_PROTECTION Protection group delete message from MSTP to NSM
- NSM_MSG_G8031_CREATE_VLAN_GROUP Messages from G8031-EPS for VLAN creation
- NSM_MSG_G8032_CREATE_VLAN_GROUP Messages from G8032-EPS for VLAN creation
- NSM_MSG_BRIDGE_DEL_G8032_VLAN Messages from G8032-EPS for VLAN deletion
- NSM_MSG_ELMI_AUTO_VLAN_ADD_PORT Messages for Provider Edge Port
- NSM_MSG_ELMI_AUTO_VLAN_DEL_PORT Messages for provider Edge Port
- NSM_MSG_VLAN_PORT_BULK Messages for list of interfaces where this vid is configured.

Data Structure

The following is the common data structure that is used with all the VLAN protocol modules.

nsm_cvlan_reg_tab

The following data structure is defined in `nsm/L2/nsm_pro_vlan.h`.

Type	Definition
`port_list`	Each instance has its own port list. The IST has all ports added to it.
`reg_tab`	Holds the VLAN information (of type avl_tree).
`bridge`	Holds bridge related information (of type nsm_bridge).

Type	Definition
svlan_tree	Holds the static VLAN information (of type avl_tree) name.
name	Reg tab name.

Definition

```
struct nsm_cvlan_reg_tab
{
#define NSM_CVLAN_REG_TAB_NAME_MAX 16
  struct list  *port_list;
  struct avl_tree *reg_tab;
  struct nsm_bridge *bridge;
  struct avl_tree *svlan_tree;
  char name [NSM_CVLAN_REG_TAB_NAME_MAX + 1];
}
```

API Definitions

The API calls in this chapter are for use with VLANs. It includes the following API calls:

API Function	Description
nsm_all_vlan_show	Shows the VLAN characteristics for VLANs on a bridge.
nsm_cvlan_reg_tab_delete	Deletes a CVLAN registration table from a bridge.
nsm_cvlan_reg_tab_entry_add	Adds a CVLAN registration entry to a CVLAN registration table.
nsm_cvlan_reg_tab_entry_delete	Deletes a CVLAN registration table entry from the CVLAN registration table.
nsm_cvlan_reg_tab_entry_delete_by_svid	Deletes a CVLAN registration table entry from the CVLAN registration table, based on the SVLAN ID.
nsm_cvlan_reg_tab_get	Adds a CVLAN registration table to a bridge.
nsm_cvlan_reg_tab_if_apply	Adds a CVLAN registration table to a customer edge (CE) port.
nsm_cvlan_reg_tab_if_delete	Removes a CVLAN registration table from a CE port.
nsm_pvlan_associate	Associates a secondary VLAN to a primary VLAN.
nsm_pvlan_associate_clear	Removes the association of secondary VLAN from a primary VLAN.
nsm_pvlan_associate_clear_all	Removes an association from all secondary VLANs.
nsm_pvlan_api_clear_port_mode	Removes the private VLAN mode from a port.
nsm_pvlan_api_host_association	Associates a primary and secondary VLAN to a host port.

API Function	Description
nsm_pvlan_api_host_association_clear_all	Clears all primary/secondary VLAN configurations of a host port.
nsm_pvlan_api_set_port_mode	Sets the private VLAN mode for an interface.
nsm_pvlan_api_switchport_mapping	Associates a primary VLAN and secondary VLAN to a promiscuous port.
nsm_pvlan_api_switchport_mapping_clear	Removes association of a secondary VLAN from a primary VLAN for a given interface.
nsm_pvlan_api_switchport_mapping_clear_all	Removes all secondary-to-primary VLAN associations from a given interface.
nsm_pvlan_configure	Configures a VLAN as a private VLAN.
nsm_pvlan_configure_clear	Removes the configuration from a private VLAN.
nsm_vlan_add	Adds a VLAN.
nsm_vlan_add_all_except_vid	Adds all VLANs to a trunk/hybrid port.
nsm_vlan_add_hybrid_port	Adds a VLAN to a hybrid port.
nsm_vlan_add_provider_port	Adds a VLAN to a provider port.
nsm_vlan_add_trunk_port	Adds a VLAN to a trunk port.
nsm_vlan_clear_hybrid_port	Resets the port mode from hybrid to access.
nsm_vlan_clear_trunk_port	Resets the port mode from trunk to access.
nsm_vlan_config_write	Shows the VLAN configuration for all bridges.
nsm_vlan_delete	Deletes a VLAN.
nsm_vlan_delete_hybrid_port	Removes a VLAN from a hybrid port.
nsm_vlan_delete_provider_port	Deletes a VLAN from a provider port.
nsm_vlan_delete_trunk_port	Removes VLAN from a trunk port.
nsm_vlan_if_config_write	Shows the VLAN configuration for an interface.
nsm_vlan_set_acceptable_frame_type	Sets acceptable frame types for the port.
nsm_vlan_set_access_port	Sets the default VLAN for an access port.
nsm_vlan_set_ingress_filter	Sets the ingress-filtering characteristic for the port.
nsm_vlan_set_hybrid_port	Sets the default VLAN for a hybrid port.
nsm_vlan_set_mtu	Sets the MTU for a VLAN.
nsm_vlan_set_native_vlan	Sets the native (default) VLAN for a trunk port.
nsm_vlan_set_provider_port	Sets the default VLAN for a provider port.

API Function	Description
nsm_vlan_trans_tab_entry_add	Adds an SVLAN translation table entry.
nsm_vlan_trans_tab_entry_delete	Deletes an SVLAN translation table entry.

nsm_all_vlan_show

This function shows the VLAN characteristics for VLANs on a bridge.

Syntax

```
void
nsm_all_vlan_show (struct cli *cli, char *brname,
                   int type, nsm_vid_t vid)
```

Input Parameters

cli	CLI pointer.
brname	Bridge name.
type	Type of VLANs.
vid	VLAN ID to be configured.

Output Parameters

None

Return Values

None

nsm_cvlan_reg_tab_delete

This function deletes a CVLAN registration table from a bridge.

Syntax

```
void
nsm_cvlan_reg_tab_delete (struct nsm_bridge *bridge,
                          char *cvlan_reg_tab_name)
```

Input Parameters

bridge	Bridge from which the CVLAN registration table is to be deleted.
cvlan_reg_tab_name	Name of the CVLAN registration table.

Output Parameters

None

Return Values

None

nsm_cvlan_reg_tab_entry_add

This function adds a CVLAN registration entry to a CVLAN registration table.

Syntax

```
s_int32_t
nsm_cvlan_reg_tab_entry_add (struct nsm_cvlan_reg_tab *regtab, u_int16_t cvid,
                             u_int16_t svid)
```

Input Parameters

regtab	The registration table to which the CVLAN registration table entry is to be added.
cvid	CVLAN ID to be translated.
svid	SVLAN ID to be translated.

Output Parameters

None

Return Values

NSM_PRO_VLAN_ERR_CVLAN_PORT_NOT_FOUND

NSM_PRO_VLAN_ERR_CVLAN_MAP_ERR

NSM_PRO_VLAN_ERR_CVLAN_MAP_EXISTS

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_NOT_EDGE_BRIDGE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_VLAN_ERR_SVLAN_NOT_FOUND

NSM_L2_ERR_MEM

NSM_PRO_VLAN_ERR_CVLAN_REGIS_HAL_ERR

nsm_cvlan_reg_tab_entry_delete

This function deletes a CVLAN registration table entry from the CVLAN registration table.

Syntax

```
s_int32_t
nsm_cvlan_reg_tab_entry_delete (struct nsm_cvlan_reg_tab *regtab,
                                u_int16_t cvid)
```

Input Parameters

regtab	The registration table from which the CVLAN registration table entry is to be deleted.
cvid	CVLAN ID of the registration table entry to be deleted.

Output Parameters

None

Return Values

0 when function succeeds

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PRO_VLAN_ERR_CVLAN_MAP_ERR

NSM_VLAN_ERR_SVLAN_NOT_FOUND

NSM_L2_ERR_NONE When function succeeds

nsm_cvlan_reg_tab_entry_delete_by_svid

This function deletes a CVLAN registration table entry from the CVLAN registration table, based on the SVLAN ID.

Syntax

```
s_int32_t
nsm_cvlan_reg_tab_entry_delete_by_svid (struct nsm_cvlan_reg_tab *regtab,
                                        u_int16_t svid)
```

Input Parameters

`regtab`	The registration table from which the CVLAN registration table entry is to be deleted.
`svid`	SVLAN ID for which the all corresponding registration table entries are to be deleted.

Output Parameters

None

Return Values

NSM_L2_ERR_NONE When the function succeeds

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

nsm_cvlan_reg_tab_get

This function adds a CVLAN registration table to a bridge.

Syntax

```
struct nsm_cvlan_reg_tab *
nsm_cvlan_reg_tab_get (struct nsm_bridge *bridge,
                       char *cvlan_reg_tab_name)
```

Input Parameters

`bridge`	Bridge to which the CVLAN registration table is to be added.
`cvlan_reg_tab_name`	Name of the CVLAN registration table.

Output Parameters

None

Return Values

Returns registration table when function success.

NULL when function fails.

nsm_cvlan_reg_tab_if_apply

This function applies a CVLAN registration table to a customer edge port.

Syntax

```
s_int32_t
nsm_cvlan_reg_tab_if_apply (struct interface *ifp,
                            char *cvlan_reg_tab_name)
```

Input Parameters

`ifp` Interface pointer.

`cvlan_reg_tab_name`

Name of the CVLAN registration table.

Output Parameters

None

Return Values

NSM_L2_ERR_NONE when the function succeeds

NSM_L2_ERR_INVALID_ARG

NSM_PRO_VLAN_ERR_CVLAN_REGIS_EXISTS

NSM_PRO_VLAN_ERR_INVALID_MODE

NSM_PRO_VLAN_ERR_CVLAN_REGIS_NOT_FOUND

NSM_PRO_VLAN_ERR_CVLAN_MAP_ERR

NSM_ERR_CROSSED_MAX_EVC

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_PRO_VLAN_ERR_CVLAN_REGIS_HAL_ERR

nsm_cvlan_reg_tab_if_delete

This function removes a CVLAN registration table from a customer edge port.

Syntax

```
s_int32_t
nsm_cvlan_reg_tab_if_delete (struct interface *ifp)
```

Input Parameters

`ifp` Interface pointer.

Output Parameters

None

Return Values

NSM_L2_ERR_NONE When function succeeds

NSM_L2_ERR_INVALID_ARG

NSM_PRO_VLAN_ERR_CVLAN_REGIS_NOT_FOUND

NSM_PRO_VLAN_ERR_INVALID_MODE

nsm_pvlan_associate

This function associates a secondary VLAN to a primary VLAN.

Syntax

```
int
nsm_pvlan_associate (struct nsm_bridge_master *master, char *brname,
                     u_int16_t vid, u_int16_t pvid)
```

Input Parameters

`master`	Reference to the NSM bridge master structure.
`brname`	Name of the bridge.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

Return Values

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_NO_VLAN_CONFIGURED

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_PVLAN_ERR_PRIMARY_SECOND_SAME

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_NOT_SECONDARY_VLAN

NSM_PVLAN_ERR_ISOLATED_VLAN_EXISTS

NSM_PVLAN_ERR_ASSOCIATED_TO_PRIMARY

nsm_pvlan_associate_clear

This function removes the association of a secondary VLAN from a primary VLAN.

Syntax

```
int
nsm_pvlan_associate_clear (struct nsm_bridge_master *master, char *brname,
                           u_int16_t vid, u_int16_t pvid)
```

Input Parameters

`master`	Reference to the NSM bridge master structure.
`brname`	Name of the bridge.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

Return Values

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_NO_VLAN_CONFIGURED

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_NOT_SECONDARY_VLAN

nsm_pvlan_associate_clear_all

This function removes an association of all secondary VLANs from a primary VLAN.

Syntax

```
int
nsm_pvlan_associate_clear_all (struct nsm_bridge_master *master,
                               char *brname, u_int16_t vid)
```

Input Parameters

`master`	Reference to the NSM bridge master structure.
`brname`	Name of the bridge.
`vid`	Primary VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_NO_VLAN_CONFIGURED

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

nsm_pvlan_api_clear_port_mode

This function removes the private VLAN mode (host or promiscuous) from a port.

Syntax

```
int
nsm_pvlan_api_clear_port_mode (struct interface *ifp,
                               enum nsm_pvlan_port_mode mode)
```

Input Parameters

`ifp`	Interface pointer.
`mode`	Mode (host or promiscuous).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_INVALID_MODE

nsm_pvlan_api_host_association

This function associates a primary and secondary VLAN to a host port.

Syntax

```
int
nsm_pvlan_api_host_association (struct interface *ifp, u_int16_t vid,
                                u_int16_t pvid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_VLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_NOT_HOST_PORT

NSM_NO_VLAN_CONFIGURED

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_SECOND_NOT_ASSOCIATED

nsm_pvlan_api_host_association_clear

This function clears the association of primary and secondary VLANs of a host port.

Syntax

```
int
nsm_pvlan_api_host_association_clear (struct interface *ifp, u_int16_t vid,
                                      u_int16_t pvid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_SECOND_NOT_ASSOCIATED

nsm_pvlan_api_host_association_clear_all

This function clears all primary and secondary VLAN configurations of a host port.

Syntax

```
int
nsm_pvlan_api_host_association_clear_all (struct interface *ifp)
```

Input Parameters

`ifp`	Interface pointer.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_PVLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_SECONDARY_VLAN

nsm_pvlan_api_set_port_mode

This function sets the private VLAN mode (host or promiscuous) for an interface.

Syntax

```
int
nsm_pvlan_api_set_port_mode (struct interface *ifp,
                             enum nsm_pvlan_port_mode mode)
```

Input Parameters

`ifp`	Interface pointer.
`mode`	Mode (host or promiscuous).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_INVALID_MODE

RESULT_ERROR -1

NSM_PVLAN_ERR_INVALID_MODE

nsm_pvlan_api_switchport_mapping

This function associates a primary VLAN and secondary VLAN to a promiscuous port.

Syntax

```
int
nsm_pvlan_api_switchport_mapping (struct interface *ifp, u_int16_t vid,
                                  u_int16_t pvid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_VLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_INVALID_MODE

NSM_NO_VLAN_CONFIGURED

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_SECOND_NOT_ASSOCIATED

nsm_pvlan_api_switchport_mapping_clear

This function removes an association of a secondary VLAN from a primary VLAN for a given interface.

Syntax

```
int
nsm_pvlan_api_switchport_mapping_clear (struct interface *ifp, u_int16_t vid,
                                        u_int16_t pvid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`pvid`	Secondary VLAN ID to be configured.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_VLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_INVALID_MODE

NSM_NO_VLAN_CONFIGURED

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

NSM_PVLAN_ERR_INVALID_MODE

NSM_PVLAN_ERR_SECOND_NOT_ASSOCIATED

NSM_VLAN_ERR_VLAN_NOT_FOUND

nsm_pvlan_api_switchport_mapping_clear_all

This function removes all secondary-to-primary VLAN associations from a given interface.

Syntax

```
int
nsm_pvlan_api_switchport_mapping_clear_all (struct interface *ifp)
```

Input Parameters

`ifp`	Interface pointer.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_PVLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

NSM_PVLAN_ERR_NOT_PRIMARY_VLAN

nsm_pvlan_configure

This function configures a VLAN as a private VLAN. The private VLAN type is specified to select either primary, isolated, or community.

Syntax

```
int
nsm_pvlan_configure (struct nsm_bridge_master *master, char *brname,
                     u_int16_t vid, enum nsm_pvlan_type type)
```

Input Parameters

`master`	Reference to the NSM bridge master structure.
`brname`	Name of the bridge.
`vid`	VLAN ID to be configured.
`type`	Private VLAN type (community, isolated, primary or none).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_NO_VLAN_CONFIGURED

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_VLAN_NOT_FOUND

nsm_pvlan_configure_clear

This function removes the configuration from a private VLAN.

Syntax

```
int
nsm_pvlan_configure_clear (struct nsm_bridge_master *master, char *brname,
                           u_int16_t vid, u_char type)
```

Input Parameters

`master`	Reference to the NSM bridge master structure.
`brname`	Name of the bridge.

`vid`	VLAN ID to be configured.
`type`	Private VLAN type (community, isolated, primary or none).

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN

NSM_NO_VLAN_CONFIGURED when no VLAN is configured

NSM_PVLAN_ERR_PROVIDER_BRIDGE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_PVLAN_ERR_NOT_CONFIGURED

nsm_vlan_add

This function adds a VLAN.

Syntax

```
int
nsm_vlan_add (struct nsm_bridge_master *master, char *brname, char *vlan_name,
              u_int16_t vid, enum nsm_vlan_state state, u_int16_t type)
```

Input Parameters

`master`	Bridge master.
`brname`	Bridge name.
`vlan_name`	VLAN name.
`vid`	VLAN ID to be configured.
`state`	VLAN state.
`type`	VLAN type.

Output Parameters

None

Return Values

0 when function succeeds.

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE

when a bridge is not aware of VLAN

NSM_VLAN_ERR_NOMEM

NSM_VLAN_ERR_RESERVED_IN_HW

NSM_VLAN_ERR_NOT_EDGE_BRIDGE

NSM_VLAN_ERR_NOT_PROVIDER_BRIDGE

NSM_VLAN_ERR_GENERAL when generic error occurs

nsm_vlan_add_all_except_vid

This function adds all VLANs, except the specified VLANs, to a trunk/hybrid port.

Syntax

```
int
nsm_vlan_add_all_except_vid (struct interface *ifp, enum nsm_vlan_port_mode
                             mode, enum nsm_vlan_port_mode sub_mode,
                             struct nsm_vlan_bmp *excludeBmp,
                             enum nsm_vlan_egress_type egress_tagged,
                             bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`mode`	VLAN port mode (trunk/hybrid).
`sub_mode`	VLAN port sub mode. Applicable for ports in the provider edge bridge (access, trunk, or hybrid).
`excludeBmp`	Bitmap of VLANs to be excluded.
`egress_tagged`	Flag to set port egress tagged/untagged for the VLAN.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_GENERAL when generic error occurs

nsm_vlan_add_hybrid_port

This function adds a VLAN to a hybrid port.

Syntax

```
int
nsm_vlan_add_hybrid_port (struct interface *ifp, nsm_vid_t vid,
                          enum nsm_vlan_egress_type egresstagged,
                          bool_t iterate_members, bool_t notify_kernel)
```


Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`egresstagged`	Flag to set port egress tagged/untagged for the VLAN.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_VLAN_ERR_CONFIG_PVID_TAG

nsm_vlan_add_provider_port

This function adds a VLAN to a provider port.

Syntax

```
int
nsm_vlan_add_provider_port (struct interface *ifp, nsm_vid_t vid,
                            enum nsm_vlan_port_mode mode,
                            enum nsm_vlan_port_mode sub_mode,
                            enum nsm_vlan_egress_type egresstagged,
                            bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`egresstagged`	Flag to set port egress tagged/untagged for the VLAN
`mode`	VLAN port mode (customer-edge, customer-network, provider-network, trunk, access or hybrid).
`sub_mode`	VLAN port sub mode. Applicable for ports in the provider edge bridge (access, trunk, or hybrid).
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_CVLAN_PORT_REG_EXIST

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_NATIVE_VID

nsm_vlan_add_trunk_port

This function adds a VLAN to a trunk port.

Syntax

```
int nsm_vlan_add_trunk_port (struct interface *ifp, nsm_vid_t vid)
bool_t iterate_members, bool_t notify_kernel);
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_INVALID_MODE

nsm_vlan_clear_hybrid_port

This function resets the port mode from hybrid to access.

Syntax

```
int
nsm_vlan_clear_hybrid_port (struct interface *ifp, bool_t iterate_members,
                            bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.

`notify_kernel` Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

nsm_vlan_clear_trunk_port

This function resets the port mode from trunk to access.

Syntax

```
int
nsm_vlan_clear_trunk_port (struct interface *ifp, bool_t iterate_members,
                           bool_t notify_kernel)
```

Input Parameters

`ifp` Interface pointer.

`iterate_members` Will be set to 1 = PAL_TRUE when the command is given for an aggregator.

`notify_kernel` Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found

NSM_VLAN_ERR_INVALID_MODE

nsm_vlan_config_write

This function shows the VLAN configuration for all bridges.

Syntax

```
int
nsm_vlan_config_write (struct cli *cli)
```

Input Parameters

`cli` CLI pointer.

Output Parameters

None

Return Values

Returns 0 when generic failure occurs.

Returns Number of lines written when success.

nsm_vlan_delete

This function deletes a VLAN.

Syntax

```
int
nsm_vlan_delete (struct nsm_bridge_master *master, char *brname,
                 u_int16_t vid, u_int16_t type, bool_t notify_pm)
```

Input Parameters

`master`	Bridge master.
`brname`	Bridge name.
`vid`	VLAN ID to be configured.
`type`	VLAN type.
`notify_pm`	Notify listeners with a private message.

Output Parameters

None

Return Values

0 when function succeeds.

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE

when bridge is not aware of VLAN NSM_VLAN_ERR_NOMEM

NSM_VLAN_ERR_VLAN_NOT_FOUND when a specified VLAN is not found

NSM_VLAN_ERR_G8031_CONFIG_EXIST

NSM_VLAN_ERR_G8032_CONFIG_EXIST

NSM_VLAN_ERR_NOT_PROVIDER_BRIDGE

NSM_VLAN_ERR_GENERAL when generic error occurs

NSM_VLAN_ERR_CVLAN_REG_EXIST

NSM_VLAN_ERR_VLAN_NOT_FOUND

nsm_vlan_delete_hybrid_port

This function removes a VLAN from a hybrid port.

Syntax

```
int
nsm_vlan_delete_hybrid_port (struct interface *ifp, nsm_vid_t vid,
```



```
                                    bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be removed.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_NATIVE_VID

nsm_vlan_delete_provider_port

This function deletes a VLAN from a provider port.

Syntax

```
int
nsm_vlan_delete_provider_port (struct interface *ifp, nsm_vid_t vid,
                               enum nsm_vlan_port_mode mode,
                               enum nsm_vlan_port_mode sub_mode,
                               bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`mode`	VLAN port mode (CE, CN, provider-network, trunk, access or hybrid).
`sub_mode`	VLAN port sub mode. Applicable for ports in the PE bridge (access, trunk, or hybrid).
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_CVLAN_PORT_REG_EXIST

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_NATIVE_VID

nsm_vlan_delete_trunk_port

This function removes VLAN from a trunk port. This function starts `switchport trunk allowed vlan remove`.

Syntax

```
int
nsm_vlan_delete_trunk_port (struct interface *ifp, nsm_vid_t vid,
                            bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_CVLAN_PORT_REG_EXIST

NSM_VLAN_ERR_IFP_NOT_BOUND when nsm local interface is not found

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_NATIVE_VID

nsm_vlan_if_config_write

This function shows the VLAN configuration for an interface.

Syntax

```
int
nsm_vlan_if_config_write (struct cli *cli, struct interface *ifp)
```

Input Parameters

`cli`	CLI pointer.
`ifp`	Interface pointer.

Output Parameters

None

Return Values

-1 when function fails.

0 when function succeeds No return codes in this function (its -1 on failure and 0 on success).

nsm_vlan_set_acceptable_frame_type

This function sets acceptable frame types for the port.

Syntax

```
int
nsm_vlan_set_acceptable_frame_type (struct interface *ifp,
                                    enum nsm_vlan_port_mode mode,
                                    int acceptable_frame_type)
```

Input Parameters

`ifp`	Interface pointer.
`mode`	VLAN port mode (trunk, hybrid, or access).
`acceptable_frame_type`	Acceptable frames.

Output Parameters

None

Return Values

-1 (ERROR) when function fails. (error found in configuration)

0 when function succeeds.

Return value from HAL (if is HAL used).

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found.

nsm_vlan_set_access_port

This function sets the default VLAN for an access port.

Syntax

```
int
nsm_vlan_set_access_port (struct interface *ifp, nsm_vid_t pvid,
                          bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`pvid`	Port default VLAN ID to be configured.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.

notify_kernel	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_VLAN_NOT_FOUND

NSM_VLAN_ERR_BASE

nsm_vlan_set_ingress_filter

This function sets the ingress-filtering characteristic for the port.

Syntax

```
int
nsm_vlan_set_ingress_filter (struct interface *ifp,
                             enum nsm_vlan_port_mode mode,
                             enum nsm_vlan_port_mode sub_mode,
                             int enable)
```

Input Parameters

ifp	Interface pointer.
mode	VLAN port mode (CE, CN, provider-network, trunk, access or hybrid).
sub_mode	VLAN port sub mode. Applicable for ports in the PE bridge (access, trunk, or hybrid).
enable	Ingress filter enable/disable flag.

Output Parameters

None

Return Values

-1 (ERROR) when function fails.

0 when function succeeds.

nsm_vlan_set_hybrid_port

This function sets the default VLAN for a hybrid port.

Syntax

```
int
nsm_vlan_set_hybrid_port (struct interface *ifp, nsm_vid_t pvid,
                          bool_t iterate_members, bool_t notify_kernel)
```


Input Parameters

`ifp`	Interface pointer.
`pvid`	Port default VLAN ID to be configured.
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

NSM_VLAN_ERR_CVLAN_PORT_REG_EXIST

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_NATIVE_VID

nsm_vlan_set_mtu

This function sets the MTU for a VLAN.

Syntax

```
int
nsm_vlan_set_mtu (struct nsm_bridge_master *master, char *brname,
                  u_int16_t vid, u_char vlan_type, u_int32_t mtu_val)
```

Input Parameters

`master`	Bridge master.
`brname`	Bridge name.
`vid`	VLAN ID to be configured.
`vlan_type`	VLAN type.
`mtu_val`	VLAN MTU.

Output Parameters

None

Return Values

NSM_VLAN_ERR_BRIDGE_NOT_FOUND when a specified bridge is not found.

NSM_VLAN_ERR_BRIDGE_NOT_VLAN_AWARE when bridge is not aware of VLAN.

NSM_VLAN_ERR_VLAN_NOT_FOUND when a specified VLAN is not found.

NSM_VLAN_ERR_GENERAL when generic error occurs.

0 when function succeeds.

nsm_vlan_set_native_vlan

This function sets the native (default) VLAN for a trunk port.

Syntax

```
int
nsm_vlan_set_native_vlan (struct interface *ifp, nsm_vid_t native_vid)
```

Input Parameters

`ifp`	Interface pointer.
`native_vid`	VLAN ID to be configured.

Output Parameters

None

Return Values

0 when function succeeds.

NSM_VLAN_ERR_IFP_NOT_BOUND when NSM local interface is not found

NSM_VLAN_ERR_INVALID_MODE

NSM_VLAN_ERR_GENERAL when generic error occurs

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_VLAN_ERR_VLAN_NOT_FOUND

nsm_vlan_set_provider_port

This function sets the default VLAN for a provider port.

Syntax

```
int
nsm_vlan_set_provider_port (struct interface *ifp, nsm_vid_t pvid,
                            enum nsm_vlan_port_mode mode,
                            enum nsm_vlan_port_mode sub_mode,
                            bool_t iterate_members, bool_t notify_kernel)
```

Input Parameters

`ifp`	Interface pointer.
`pvid`	Port default VLAN ID to be configured.
`mode`	VLAN port mode (CE, CN provider-network, trunk, access or hybrid).
`sub_mode`	VLAN port sub mode. Applicable for ports in the PE bridge (access, trunk, or hybrid).
`iterate_members`	Will be set to 1 = PAL_TRUE when the command is given for an aggregator.
`notify_kernel`	Will be set to true when the HAL call has to be called for the configuration.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

nsm_vlan_trans_tab_entry_add

This function adds an SVLAN translation table entry.

Syntax

```
s_int32_t
nsm_vlan_trans_tab_entry_add (struct interface *ifp, u_int16_t vid,
                              u_int16_t trans_vid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.
`trans_vid`	The translated VLAN ID.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_L2_ERR_NONE When function succeeds

NSM_L2_ERR_INVALID_ARG

NSM_VLAN_ERR_VLAN_NOT_IN_PORT

NSM_PRO_VLAN_ERR_VLAN_TRANS_EXISTS

NSM_L2_ERR_MEM

nsm_vlan_trans_tab_entry_delete

This function deletes an SVLAN translation table entry.

Syntax

```
s_int32_t
nsm_vlan_trans_tab_entry_delete (struct interface *ifp, u_int16_t vid)
```

Input Parameters

`ifp`	Interface pointer.
`vid`	VLAN ID to be configured.

Output Parameters

None

Return Values

0 when function succeeds.

Negative value (ERROR) when function fails.

NSM_L2_ERR_NONE When function succeeds

NSM_L2_ERR_INVALID_ARG

CHAPTER 8 VLAN Registration Protocol

This chapter describes the VLAN registration protocol (VRP) implementation in ZebOS-XP, including both Multiple VLAN Registration Protocol (MVRP) and GARP VLAN Registration Protocol (GVRP). This section includes an overview, a list of features, and a description of the supported functions.

Overview

ZebOS-XP supports the following types of VLAN registration protocols:

- Multiple VLAN Registration Protocol: MVRP registers multiple VLANs and provides for the rapid recovery of network failures without interrupting services to unaffected VLANs. In addition, MVRP improves the convergence time of the GVRP module.
- GARP VLAN Registration Protocol: GVRP provides support for 802.1Q VLAN pruning and dynamic VLAN creation. A switch can be used to exchange VLAN configuration information with other GVRP switches, prune unnecessary broadcast and unknown unicast traffic, and dynamically create and manage VLANs.

Note: This section has MVRP and GVRP information. For additional VLAN information, refer to Chapter 7, *Virtual LANs*.

Architecture

MVRP is the successor to GVRP. MVRP is a standards-based Layer 2 network protocol that is used to configure automatically VLAN information on switches. Refer to the 802.1ak amendment to 802.1Q-2005 for the complete definition of MVRP.

MVRP provides the mechanism to dynamically share VLAN information and configure the required VLANs within a Layer 2 network. For example, when using MVRP, users only have to configure an end port or a VLAN-supported device connected to a switchport when adding a switch port to a VLAN. After this is configured, all necessary VLAN trunks are dynamically created on the other MVRP-enabled switches. Without using MVRP, the user has to configure manually a VLAN trunk. In addition, MVRP updates dynamic VLAN entries in the filtering database. This helps maintain VLAN configuration dynamically based on the current network configuration.

By definition, MVRP is an MRP application. It is designed to help provide VLAN registration services. MVRP utilizes MRP Attribute Declaration (MAD) and MRP Attribute Propagation (MAP) to provide a common state machine description and a common information propagation mechanism, both of which are defined for use in each MRP-based application. In addition, MVRP includes a mechanism for dynamic maintenance of the contents of dynamic VLAN registration entries for VLANs. This helps propagate VLAN information to other bridges. This information allows MVRP-aware devices to dynamically establish and update their VLAN sets that currently have active members. All of this functionality provided by MVRP allows switches to discover automatically VLAN information that might otherwise need to be configured manually.

Data Structures

The following are the common data structures that are used with the GVRP module.

garp

This structure contains the list of function pointers to the various activities of join or leave indication, propagation, PDU transmit, VID of the corresponding application (MMRP or MVRP). The following data structure is defined in nsm/L2/garp/garp.h.

Members	Definition
`lib_globals *garpm`	Pointer to the management control structure for GARP applications (GVRP and GMRP).
`reg_proto_type proto`	Callback GARP application function that gets up-called when a join indication is received for a GARP application.
`gid *gid_instance`	Callback GARP application function that gets up-called when leave indication is received for the GARP application.
`Join_indication_func`	Callback function that is called when join indication is received.
`leave_indication_func`	Callback function that is called when leave indication is received.
`join_propagated_func`	Callback function that is called when join indication is received.
`leave_propagated_func`	Callback function that is called when leave indication is received.
`transmit_func`	Callback function is called when gid instance associated with the port requires to transmit garp pdu.
`get_bridge_instance_func`	Callback function for bridge instance function.
`get_vid_func`	Callback function for getting VID.
`get_svid_func`	Callback function for getting SVID.
`get_gid_func`	Callback function for getting gid instance.

Definition

```
struct garp {
  /* pointer to the management control structure for the garp application
    (GVRP, GMRP) */
  struct lib_globals *garpm;
  enum reg_proto_type proto;
  /* Callback garp application function that gets upcalled when join
     indication is received for the garp application */
  void(*join_indication_func) (void *application,
                               struct gid *gid_instance,
                               u_int32_t joining_gid_index);
  /* Callback garp application function that gets upcalled when leave
```

```
     indication is received for the garp application */
  void(*leave_indication_func) (void *application,
                                struct gid *gid_instance,
                                u_int32_t leaving_gid_index);
  /* Callback garp application function that gets upcalled when join
     indication is received for the garp application */
  void(*join_propagated_func) (void *application,
                               struct gid *gid_instance,
                               u_int32_t joining_gid_index);
 /* Callback garp application function that gets upcalled when join
     indication is received for the garp application */
  void(*join_propagated_func) (void *application,
                               struct gid *gid_instance,
                               u_int32_t joining_gid_index);
  /* Callback garp application function that gets upcalled when leave
     indication is received for the garp application */
  void(*leave_propagated_func) (void *application,
                                struct gid *gid_instance,
                                u_int32_t leaving_gid_index);
  /* Callback garp application function that gets upcalled when a gid_instance
     associated with the port requires to transmit a garp pdu */
  bool_t (*transmit_func) (void *application, struct gid *gid_instance);
  void* (*get_bridge_instance_func) (void *application);
  u_int16_t (*get_vid_func) (void *application);
  u_int16_t (*get_svid_func) (void *application);
  struct gid* (*get_gid_func)(const void *port, u_int16_t vid, u_int16_t
svid);
};
```

garp_instance

This structure maintains both applications (MMRP and MVRP) that contain the application of this instance and the count of the number of packets transmitted and received on this instance. The following data structure is defined in nsm/L2/garp/garp.h.

Member	Definition
Garp	Holds the garp information (of type garp).
Application	Holds the garp application (GMRP, GVRP) information (of type garp).
Gip_table	Holds garp information propagation instance (of type ptree).
Receive_counters	Holds the count of number of packets received.
Transmit_counters	Holds the count of number of packets transmitted.

Definition

```
struct garp_instance {
  /*Pointer to GARP */
  struct garp *garp;
  /* pointer to the garp application (GMRP, GVRP) */
  struct garp *application;
  /* pointer to garp information propagation instance */
```

```
  struct ptree *gip_table;
  /* num_pkts, leave_all, join_empty, join_in, leave_empty, leave_in, empty */
#if defined HAVE_MMRP || defined HAVE_MVRP
  unsigned int receive_counters[MRP_ATTR_EVENT_MAX + 1];
#else
  unsigned int receive_counters[GARP_ATTR_EVENT_MAX + 1];
#endif /* HAVE_MMRP || HAVE_MVRP */

  /* num_pkts, leave_all, join_empty, join_in, leave_empty, leave_in, empty */
#if defined HAVE_MMRP || defined HAVE_MVRP
  unsigned int transmit_counters[MRP_ATTR_EVENT_MAX + 1];
#else
  unsigned int transmit_counters[GARP_ATTR_EVENT_MAX + 1];
#endif /* HAVE_MMRP || HAVE_MVRP */
};
```

gmrp

This structure contains the MMRP bridge structure, the MRP instance for this instance and the VLAN ID. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
gmrp_bridge *gmrp_bridge	Holds bridge information (of type gmrp_bridge).
garp_instance garp_instance	Holds garp instances information (of type garp_instance).
gmrp_gmd *gmd	Pointer to gmrp_gmd.
vlanid	VLAN identifier.
svlanid	Service VLAN identifier.
avl_tree *tree	Tree pointer of type.
mac_addr[HAL_HW_LENGTH]	Hold mac address of length HAL_HW_LENGTH.

Definition

```
struct gmrp
{
  struct gmrp_bridge *gmrp_bridge;
  struct garp_instance garp_instance;
  struct gmrp_gmd *gmd;
  u_int16_t vlanid;
  u_int16_t svlanid;
  struct avl_tree *tree;
  u_char mac_addr[HAL_HW_LENGTH];
};
```


gmrp_bridge

This structure contains the bridge structure with which the MMRP application is associated, the vlan_table of the MMRP instance, the function pointers to listener callbacks, and the GARP structure with which it is associated. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
bridge	Holds bridge information (of type nsm_bridge).
gm_vlan_table	Holds gmrp VLAN table (of type gmrp_vlan_table).
gmrp_appln	Holds the gmrp application information (of type nsm_vlan_listener).
gmrp_br_appln	Holds the gmrp bridge application information (of type nsm_bridge_listener).
garp garp	Holds garp information (of type garp).
globally_enabled	Flag to represent port is gmrp globally enabled.
gmrp_registration_type reg_type	Registration type (of type gmrp_registration_type).
gmrp_last_pdu_origin	Holds the last pdu origin (either source address or ifindex).

Definition

```
struct gmrp_bridge
{
  struct nsm_bridge *bridge;
  struct gmrp_vlan_table *gm_vlan_table;
  struct nsm_vlan_listener *gmrp_appln;
  struct nsm_bridge_listener *gmrp_br_appln;
  struct garp garp;
  char globally_enabled;
  enum gmrp_registration_type reg_type;
  int gmrp_last_pdu_origin;
};
```

gmrp_port

This structure contains the VLAND ID associated with the port for an instance of MMRP and the MAD structure. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
port	Holds the port information
gid	GARP Information Declaration: one per port/per application
flags	Flags representing the forward status: GMRP_MGMT_PORT_CONFIGURED GMRP_MGMT_FORWARD_ALL_CONFIGURED GMRP_MGMT_FORWARD_ALL_FORBIDDEN GMRP_MGMT_FORWARD_UNREGISTERED_CONFIGURED GMRP_MGMT_FORWARD_UNREGISTERED_FORBIDDEN
globally_enabled	Flag representing whether the port is gmrp globally enabled
forward_all_cfg	Flag representing whether the port is forward all configured
registration_cfg	Flag representing the port registration configuration
gid_port	Holds the gmrp instance port information (of type gid_port)
forward_unregistered_cfg	Flag representing ports unregistration configuration
gmrp_failed_registration	Flag representing gmrp failed registration
gmrp_last_pdu_origin	Holds the last PDU origin (either source mac address or ifindex)

```
struct gmrp_port
{
  void *port;
  struct gid *gid;
  u_char flags;
  char globally_enabled;
  u_char forward_all_cfg;
  u_char registration_cfg;
  struct gid_port *gid_port;
  u_char forward_unregistered_cfg;
  u_int32_t gmrp_failed_registrations;
  u_int8_t gmrp_last_pdu_origin [ETHER_ADDR_LEN];
};
```

gmrp_port_instance

This structure contains the VLAND ID associated with the port for an instance of MMRP and the MAD structure. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
vlanid	VLAN identifier
svlanid	Service VLAN identifier
gid	GARP Information Declaration: one per port/per application
tree	Pointer to avl_tree.

Definition

```
struct gmrp_port_instance
{
  u_int16_t vlanid;
  u_int16_t svlanid;
  struct gid *gid;
  struct avl_tree *tree;
};
```

gmrp_port_config

This structure contains the VLAND ID associated with the port for an instance of MMRP and the MAD structure. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
registration	Flag representing the type of registration: GID_EVENT_NORMAL_REGISTRATION GID_EVENT_FIXED_REGISTRATION GID_EVENT_FORBID_REGISTRATION GID_EVENT_RESTRICTED_GROUP_REGISTRATION
fwd_all	Flag representing either join or leave events
join_timeout	Holds the join_timeout value (Default is GID_DEFAULT_JOIN_TIME)
leave_timeout	Holds the leave timeout value
leave_all_timeout	Holds the leave all timeout value
enable_port	Flag representing whether the port is gmrp enabled or disabled
p2p	Point-to-point

Definition

```
struct gmrp_port_config
```

```
{
  u_char registration;
  u_char fwd_all;
  pal_time_t join_timeout;
  pal_time_t leave_timeout;
  pal_time_t leave_all_timeout;
  bool_t enable_port;
  bool_t enable_periodic_timer;
  u_int8_t p2p;
};
```

gmrp_vlan

This structure contains the VID associated and the count of number of packets transmitted and received on that VLAN. The following data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
vlanid	VLAN identifier.
receive_counters	Hold the number of packet received.
transmit_counters	Holds the number of packet transmitted.

Definition

```
struct gmrp_vlan
{
  u_int16_t vlanid;
#ifdef HAVE_MMRP
  u_int32_t receive_counters[MRP_ATTR_EVENT_MAX+1];
  u_int32_t transmit_counters[MRP_ATTR_EVENT_MAX+1];
#else
  u_int32_t receive_counters[GARP_ATTR_EVENT_MAX+1];
  u_int32_t transmit_counters[GARP_ATTR_EVENT_MAX+1];
#endif /* HAVE_MMRP */
};
```

gmrp_vlan_table

This structure contains the array of VLANs associated with the GMRP application. This data structure is defined in nsm/L2/garp/gmrp.h.

Member	Definition
gmrp_vlan	Holds the VLAN information (array of NSM_VLAN_MAX+1)

Definition

```
struct gmrp_vlan_table
{
  struct gmrp_vlan *gmrp_vlan[NSM_VLAN_MAX + 1];
};
```

API

The API functions defined in this section are for the GVRP/MVRP functionality in ZebOS-XP.

Note: MVRP functions are the same as GVRP functions.

Function	Description
gvrp_clear_all_statistics	Clears all statistics for GVRP.
gvrp_disable	Disables GVRP.
gvrp_disable_port	Disables GVRP on a port.
gvrp_dynamic_vlan_learning_set	Dynamic VLAN learning capability for GVRP per bridge.
gvrp_enable	Enables GVRP.
gvrp_enable_port	Enables GVRP on a port.
gvrp_get_per_vlan_statistics_details	Gets statistical details for GVRP per VLAN.
gvrp_set_registration	Sets registration for GVRP.
gvrp_set_timer	Sets the timer for GVRP.

gvrp_clear_all_statistics

This function clears all statistics for GVRP.

Syntax

```
int
gvrp_clear_all_statistics (struct cli *cli)
```

Input Parameters

`cli` CLI structure.

Output Parameters

None

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_disable

This function disables GVRP.

Syntax

```
int
gvrp_disable (struct nsm_bridge_master *master,
              char* bridge_name);
```

Input Parameters

`master`	Pointer to the NSM master.
`bridge_name`	Name of the bridge.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

gvrp_disable_port

This function disables GVRP on a port.

Syntax

```
int
gvrp_disable_port (struct nsm_bridge_master *master,
                   struct interface *ifp)
```

Input Parameters

`master`	Pointer to the NSM master.
`ifp`	Interface pointer.

Output Parameters

None

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_dynamic_vlan_learning_set

This function sets dynamic VLAN learning capability for GVRP per bridge.

Syntax

```
int
gvrp_dynamic_vlan_learning_set (struct nsm_bridge_master *master,
    char *br_name, bool_t vlan_learning_enable)
```

Input Parameters

`master`	Pointer to the NSM master.

`br_name` Name of the bridge.

`vlan_learning_enable`

If the value of this parameter is 1, it enables dynamic VLAN creation. If the value is 0, it disables dynamic VLAN creation.

Output Parameters

None

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_enable

This function enables GVRP.

Syntax

```
int
gvrp_enable (struct nsm_bridge_master *master,
             char* reg_type,
             const char* const bridge_name);
```

Input Parameters

`master` Pointer to the NSM master.

`bridge_name` Name of the bridge.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

gvrp_enable_port

This function enables GVRP on a port.

Syntax

```
int
gvrp_enable_port (struct nsm_bridge_master *master,
                  struct interface *ifp)
```

Input Parameters

`master` Pointer to the NSM master.

`ifp` Interface pointer.

Output Parameters

None

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_get_per_vlan_statistics_details

This function gets statistical details for GVRP per VLAN.

Syntax

```
int
gvrp_get_per_vlan_statistics_details (struct nsm_bridge_master *master,
                                      const char* const bridge_name,
                                      const u_int16_t vid,
                                      u_int32_t *receive_counters,
                                      u_int32_t *transmit_counters)
```

Input Parameters

`master`	Pointer to the NSM master.
`bridge_name`	Name of the bridge.
`vid`	VLAN ID to be configured.

Output Parameters

`receive_counter`	Number of GVRP packets received.
`transmit_counter`	Number of GVRP packets transmitted.

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_set_registration

This function sets registration for GVRP.

Syntax

```
int
gvrp_set_registration (struct nsm_bridge_master *master,
                       struct interface *ifp,
                       const u_int32_t registration_type)
```

Input Parameters

`master`	Pointer to the NSM master.
`ifp`	Interface pointer.

`registration_type`

Choose from:

GID_REG_MGMT_NORMAL
GID_REG_MGMT_FIXED
GID_REG_MGMT_FORBIDDEN

Output Parameters

None

Return Values

CLI_SUCCESS when function succeeds.

CLI_ERROR when function fails.

gvrp_set_timer

This function sets the timer for GVRP.

Syntax

```
int
gvrp_set_timer (struct nsm_bridge_master *master,
                struct interface *ifp,
                const u_int32_t timer_type,
                const pal_time_t timer_value)
```

Input Parameters

`master`	Pointer to the NSM master.
`ifp`	Interface pointer.
`timer_type`	Choose one of these defined in the enum `garp_timers` in `nsm/L2/garp/garp_gid.c`: GARP_JOIN_TIMER GARP_LEAVE_TIMER GARP_LEAVE_ALL_TIMER GARP_LEAVE_CONF_TIMER GARP_LEAVEALL_CONF_TIMER
`timer_value`	Timer value in hundredths of a second.

Output Parameters

None

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

CHAPTER 9 Port Authentication

This chapter describes Port Authentication implementation in ZebOS-XP. This section includes an overview of Port Authentication and a description of the supported Port Authentication functions.

Overview

The ZebOS-XP port authentication module provides port-based network access control to LAN devices in support of the IEEE 802.1X Port-Based Network Access Control standard. Port authentication is valuable for enterprise LANs where users are restricted to certain services or organizations. It is also useful where a LAN offers connectivity to areas of a business building accessed by the public. Ports in Media Access Control (MAC) bridges used to attach routers to a LAN can also benefit from the features of Port Authentication.

The port authentication module supplies an extension to system functionality by providing the means to prevent unauthorized access by a supplicant to restricted system services, or to prevent a supplicant from attempting to access an unauthorized system. In addition, ZebOS-XP Port Authentication can prevent an unauthorized system outside of the network from connecting to a Supplicant LAN. Port authentication is desirable.

Features

- Port Authentication Exchange (PAE) messaging between supplicant and authentication server
- Denial of access to unauthorized requests
- Support for co-located or external authentication server designation
- RADIUS functional support, including server key exchange
- Easily add or remove ports from Port Authentication management
- Enable or disable Port Authentication system-wide or on a per-port basis
- Industry-standard Command Line Interface (CLI)
- Security from unauthorized access by clients, other systems, or Bridged LANs
- Protection from unauthorized wireless users who are within a wireless access point area
- SHOW commands display summary or detailed information or statistics for an entire system or a single port
- Flexible software module extends to new ports as the system grows

Port Authentication Basics

802.1X Port Authentication is a means to authenticate devices that attempt to attach to a LAN (or, system) port. A point-to-point connection establishes once a device authenticates successfully. A connection fails if authentication fails. Port Authentication thus enhances security for systems as it protects against unauthorized access from either directly connected or wireless supplicants.

Port Authentication uses the physical access characteristics of LAN infrastructures to provide a means of authenticating and authorizing devices attached to LAN ports. There are three distinct roles in the Port Authentication process, including:

- Authenticator - the port that enforces authentication before clients can access to services accessible via that port.

- Supplicant - a port or device that wants to access services offered by the authenticator.
- Authentication Server - a server that performs authentication required to check the credentials of the supplicant on behalf of the authenticator to indicate whether the supplicant can access the services provided by the authenticator.

Using a RADIUS (Remote Authentication Dial in User Service) server is one way to authenticate supplicant requests. RADIUS servers are responsible for receiving connection requests, authenticating the user, and then returning all configuration information necessary for the client to deliver the service requested.

Authorization States

A parameter in the Port Authentication module allows administrative control over the port's authorization status. Valid values and definitions are implemented in the Port Authentication Module as follows:

- Force Authorized forces the Authenticator PAE state machine to set the status of the controlled port to authorized
- Force Unauthorized forces the Authenticator PAE state machine to set the status of the controlled port to unauthorized
- Auto allows the Authenticator PAE state machine to control the value of the port status to reflect the outcome of the authentication exchanges between the Supplicant PAE, Authenticator PAE, and the Authentication Server.

API

The API calls defined in this subsection are for the Port Authentication functionality in ZebOS-XP. It includes the following API calls:

Function	Description
auth_port_ctrl_dir_set	Sets the packet control direction
auth_port_ctrl_set	Sets the port authentication mode to authorized, unauthorized or automatic.
auth_port_ctrl_unset	Deletes a port from 802.1x management.
auth_port_initialize_set	Deletes a port from 802.1x management.
auth_port_quiet_period_set	Sets the quiet period for an interface.
auth_port_quiet_period_unset	Resets the quiet period for an interface to default.
auth_port_reauth_period_set	Sets the seconds between reauthorization attempts.
auth_port_reauth_period_unset	Resets the seconds between reauthorization attempts for an interface to default value.
auth_port_reauthentication_set	Enables reauthentication on a port.
auth_port_reauthentication_unset	Disables reauthentication on a port.
auth_port_server_timeout_set	Sets authentication server response timeout.

Function	Description
auth_port_server_timeout_unset	Resets authentication server response timeout for an interface to default value.
auth_port_supplicant_timeout_set	Sets the supplicant response timeout.
auth_port_supplicant_timeout_unset	Resets supplicant response timeout for an interface to default value
auth_port_tx_period_set	Sets the seconds between successive request id attempts.
auth_port_tx_period_unset	Resets the seconds between successive request id attempts for an interface to default value.
auth_radius_client_address_set	Sets the local radius address.
auth_radius_client_address_unset	Resets the local radius address.
auth_radius_server_address_set	Sets the radius server address.
auth_radius_server_address_unset	Resets the radius server address.
auth_radius_shared_secret_set	Sets the radius server key.
auth_radius_shared_secret_unset	Resets the radius server key.
auth_system_ctrl_set	Enables global port authentication.
auth_radius_shared_secret_set	Sets the radius server key.
auth_system_ctrl_unset	Disables global port authentication.

auth_port_ctrl_dir_set

This function sets the packet control direction to the following:

- Discard receive packets from supplicant.
- Discard receive and transmit packets from supplicant.

Syntax

```
int
auth_port_ctrl_dir_set (u_int32_t vr_id, char *name, int dir)
```

Input Parameters

`vr_id`	Virtual router ID.
`name`	Name of the port.

`dir`	Choose from the below values:
	`AUTH_CTRL_DIR_IN`
	`AUTH_CTRL_DIR_BOTH`

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC authentication already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_ctrl_set

This function sets the port authentication mode to authorized, unauthorized or automatic.

Syntax

```
int
auth_port_ctrl_set (u_int32_t vr_id, char *name, int ctrl)
```

Input Parameters

`vr_id`	Virtual router ID.
`name`	Name of the port.
`ctrl`	Choose from the below values: AUTH_PORT_CTRL_INVALID AUTH_PORT_CTRL_FORCE_UNAUTHORIZED AUTH_PORT_CTRL_FORCE_AUTHORIZED AUTH_PORT_CTRL_AUTO AUTH_PORT_CTRL_MAX

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC authentication already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_ctrl_unset

This function deletes a port from 802.1x management.

Syntax

```
int
auth_port_ctrl_unset (u_int32_t vr_id, char *name)
```

Input Parameters

`vr_id`	Virtual router ID.
`name`	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when an interface does not exist.

auth_port_initialize_set

This function adds a port to 802.1x management.

Syntax

```
int
auth_port_initialize_set (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_port_quiet_period_set

This function sets the quiet period for an interface.

Syntax

```
int
auth_port_quiet_period_set (u_int32_t vr_id, char *name, int secs)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.
secs	Quiet period in the HELD state.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_quiet_period_unset

This function resets the quiet period for an interface to default (60).

Syntax

```
int
auth_port_quiet_period_unset (u_int32_t vr_id, char *name)
```

Input Parameters

`vr_id`	Virtual router ID.
`name`	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_reauth_period_set

This function sets the seconds between reauthorization attempts.

Syntax

```
int
auth_port_reauth_period_set (u_int32_t vr_id, char *name, u_int32_t secs)
```

Input Parameters

`vr_id`	Virtual router ID.
`name`	Name of the port.
`secs`	Seconds between reauthorization attempts.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_reauth_period_unset

This function resets the seconds between reauthorization attempts for an interface to default value (3600).

Syntax

```
int
auth_port_reauth_period_unset (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_reauthentication_set

This function enables reauthentication on a port.

Syntax

```
int
auth_port_reauthentication_set (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_reauthentication_unset

This function disables reauthentication on a port.

Syntax

```
int
auth_port_reauthentication_unset (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_server_timeout_set

This function sets authentication server response timeout.

Syntax

```
int
auth_port_server_timeout_set (u_int32_t vr_id, char *name, int secs)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.
secs	Supplicant response timeout.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_server_timeout_unset

This function resets authentication server response timeout for an interface to default value (30).

Syntax

```
int
auth_port_server_timeout_unset (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

AUTH_API_SET_ERR_INVALID_VALUE when invalid value is passed.

auth_port_supplicant_timeout_set

This function sets the supplicant response timeout.

Syntax

```
int
auth_port_supplicant_timeout_set (u_int32_t vr_id, char *name, int secs)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.
secs	Supplicant response timeout.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_port_supplicant_timeout_unset

This function resets supplicant response timeout for an interface to default value (30).

Syntax

```
int
auth_port_supplicant_timeout_unset (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

AUTH_API_SET_ERR_INVALID_VALUE when secs passed is invalid.

auth_port_tx_period_set

This function sets the seconds between successive request id attempts.

Syntax

```
int
auth_port_tx_period_set (u_int32_t vr_id, char *name, int secs)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.
secs	Seconds between successive request ID attempts.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

AUTH_API_SET_ERR_INVALID_VALUE when secs passed is invalid.

auth_port_tx_period_unset

This function resets the seconds between successive request id attempts for an interface to default value (30).

Syntax

```
int
auth_port_tx_period_unset (u_int32_t vr_id, char *name)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

MAC_AUTH_EXIST when MAC already exists.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_PORT_NOT_EXIST when the interface name does not exist.

auth_radius_client_address_set

This function sets the local radius address.

Syntax

```
int
auth_radius_client_address_set (u_int32_t vr_id, char *name, int port)
```

Input Parameters

vr_id	Virtual router ID.
name	Name of the port.
port	RADIUS client port number.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_radius_client_address_unset

This function resets the local radius address.

Syntax

```
int
auth_radius_client_address_unset (u_int32_t vr_id)
```

Input Parameters

vr_id	Virtual router ID.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_radius_server_address_set

This function sets the radius server address. This is called by the CLI r`adius-server host HOSTNAME (PORT|)`

Syntax

```
int
auth_radius_server_address_set (u_int32_t vr_id, char *name, int port,
                                u_int16_t max_retry, u_int32_t timeout,
                                u_char *key)
```

Input Parameters

vr_id	Virtual router ID.
name	RADIUS server dotted IP address or hostname format.
port	RADIUS server port number.
max_retry	RADIUS server maximum retransmit attempts.
timeout	RADIUS server timeout value.
key	Shared secret among radius server and 802.1X client.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERR_HOST_NOT_CONFIGURED when memory allocation fails (unable to configure host).

auth_radius_server_address_unset

This function resets the radius server address.

Syntax

```
int
```

```
auth_radius_server_address_unset (u_int32_t vr_id, char *name, int port)
```

Input Parameters

vr_id	Virtual router ID.
name	RADIUS server dotted IP address or host name format.
port	RADIUS server port number.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

AUTH_API_SET_ERROR when generic error occurs.

auth_radius_shared_secret_set

This function sets the radius server key.

Syntax

```
int
auth_radius_shared_secret_set (u_int32_t vr_id, char *key)
```

Input Parameters

vr_id	Virtual router ID.
key	Shared secret among radius server and 802.1X client.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_radius_shared_secret_unset

This function resets the radius server key.

Syntax

```
int
auth_radius_shared_secret_unset (u_int32_t vr_id)
```

Input Parameters

vr_id	Virtual router ID.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_system_ctrl_set

This function enables global port authentication.

Syntax

```
int
auth_system_ctrl_set (u_int32_t vr_id)
```

Input Parameters

vr_id	Virtual router ID.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

auth_system_ctrl_unset

This function disables global port authentication.

Syntax

```
int
auth_system_ctrl_unset (u_int32_t vr_id)
```

Input Parameters

vr_id	Virtual router ID.

Output Parameters

None

Return Values

AUTH_API_SET_SUCCESS when function succeeds.

AUTH_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist.

CHAPTER 10 Link Aggregation

This chapter describes the link aggregation features in ZebOS-XP. With link aggregation, a network administrator can:

- Define parallel full duplex point-to-point links between two devices to use as if they were a single link. This type of link aggregation can be dynamic or static.
- Define multiple links as a resilient load sharing interconnect between *multiple* devices in two separately administered networks. This type of link aggregation is called multi-chassis link aggregation.

Link aggregation is defined by IEEE 802.1AX-REV-D3.1

Note: Link aggregation was originally specified in IEEE 802.3ad, but was later moved to IEEE 802.1AX.

Dynamic Link Aggregation

Dynamic link aggregation combines two or more interfaces (links) to form a Link Aggregation Group (LAG) to increase the bandwidth beyond the limit of any one single interface. Each switch views the LAG as a single interface. A LAG balances traffic across its member interfaces.

The example in Figure 10-1 shows three interfaces configured between the two switches:

- On switch S1, the `eth1`, `eth2`, and `eth3` interfaces are members of the LAG
- On switch S2, the `eth2`, `eth3`, and `eth4` interfaces are members of the LAG

These three links form a single logical channel between the two switches.

Note: A LAG is also called a *channel group* or *port channel*.

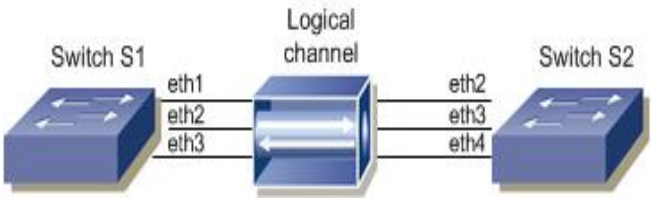

Figure 10-1: Link aggregation between switches

A LAG is fault tolerant. If a link in a LAG fails, traffic is automatically redistributed across the remaining links. This automatic recovery is transparent to network applications. When there is a failure in one physical interface, the remaining interfaces stay up, so there is no disruption to communications.

Spanning tree protocols (STP) can be used with link aggregation. STP treats a LAG as a single link and only sends bridge PDUs on one of the links. Without the use of a LAG, STP shuts down redundant links between switches until one connection goes down. This means that link aggregation can make use of all available links between two switches.

The ZebOS-XP LACP implementation supports the aggregation of up to six physical Ethernet links into a single logical channel.

Note: “EtherChannnel” is a Cisco term for a proprietary link aggregation technology that later evolved into the IEEE 802.3ad/802.1AX standards.

Link Aggregation Control Protocol

The Link Aggregation Control Protocol (LACP) provides a standard way to exchange information between partners on a link to allow them to agree on the LAG to which the link belongs and to enable its transmission and reception functions in an orderly manner.

LACP works by sending Link Aggregation Control Protocol Data Units (LACPDUs) on all links on which the protocol is enabled. If LACP finds a device on the other end of a link that also has LACP enabled, LACP sends PDUs along the same links enabling the two units to detect the multiple links between themselves and combine them into a single logical link.

LACP can be configured in one of two modes:

- In active mode, an interface initiates negotiates with remote ports by sending LACP packets
- In passive mode, an interface responds to LACP packets it receives, but does not initiate LACP negotiation

On each interface, LACP:

- Maintains configuration information
- Exchanges configuration information with other systems to allocate LAGs
- Attaches ports to and detaches ports from an aggregator when they join or leave a LAG
- Enables or disables an aggregator's collector and distributor functions

Static Link Aggregation

With static link aggregation, a network administrator must ensure that all link aggregation configuration on both participating LAG components are set up properly. LACP is *not* used to manage the configuration. This means that:

- A device cannot confirm that the configuration at the other end can handle link aggregation. A cabling or configuration mistake can go undetected.
- If a link fails, a peer system does not perceive any connectivity problems and can continue sending traffic on the link.

Either situation can lead to network problems.

Multi-Chassis Link Aggregation

This section describes the Multi-Chassis Link Aggregation implementation in ZebOS-XP. Multi-Chassis Link Aggregation is also called MC-LAG, MLAG, or Distributed Resilient Network Interconnect (DRNI). In this document, it is called MC-LAG.

Dynamic Link Aggregation binds multiple physical links in a node into a single logical link which increases bandwidth and provides link-level redundancy. However, if one of the nodes fails, the result is complete traffic loss.

MC-LAG, as specified in IEEE 802.1AX-REV-D3.1, extends the link aggregation concept to ensure that connectivity between two networks can be maintained despite the failure of a node. With MC-LAG, at either one or both ends of a link aggregation group, a single aggregation system is replaced by a *portal* that is a collection of one to three portal systems.

As shown in Figure 10-2, node 1 and node 2 share a common endpoint in node 3. Nodes 1 and 2 are a single logical node to node 3. Even if node 1 or node 2 is down, there exists a path from node 3 to reach other destinations.

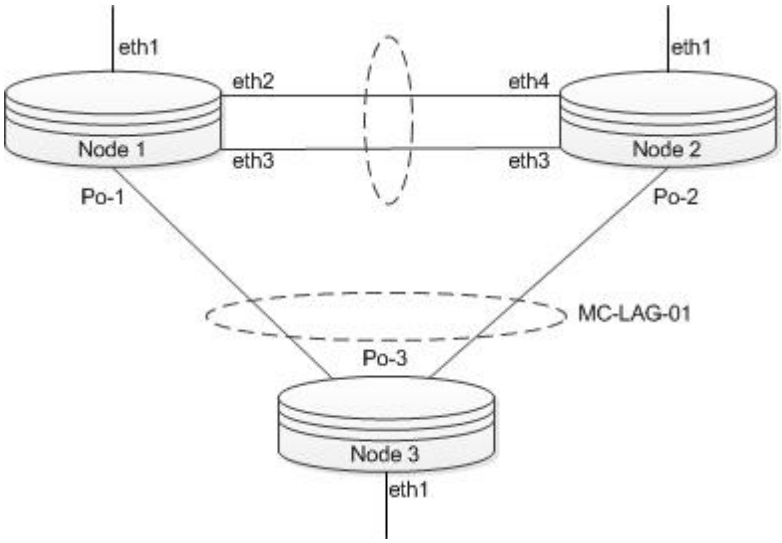

Figure 10-2: Basic MC-LAG topology

In Figure 10-2, nodes 1, 2, and 3 are a portal system, each with physical links that together make up a link aggregation group. The portal's systems cooperate to emulate the presence of a single aggregation system to which the entire link aggregation group is attached.

Figure 10-3 shows the MC-LAG architecture in a two-node portal system:

- The top of Figure 10-3 shows nodes 1 and 2 as two physically separate systems.
- The bottom of Figure 10-3 shows the nodes as one emulated system with MACs managed by a link aggregation function.
- A gateway is a virtual connection (not corresponding to a physical link) connecting a distributed relay function to a system, consisting of a gateway link and its terminating virtual MACs.
- Between the distributed relay functions in the emulated system, there is a logical link called an Intra-Portal Link (IPL), terminated at each end by an Intra-Portal Port (IPP).

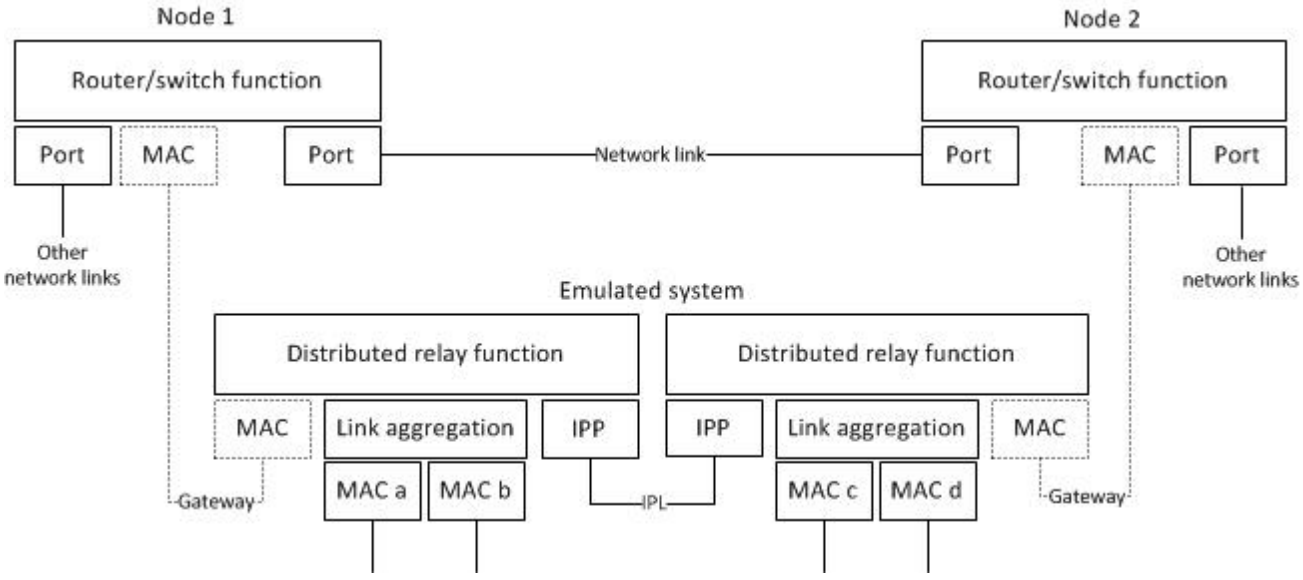

Figure 10-3: MC-LAG architecture

An emulated system in a three-node portal system has as an additional gateway port, one to each portal system, and two or three IPLs to interconnect the distributed relay functions.

MC-LAG uses the distributed relay function to interconnect two or three systems, each running link aggregation, to create a portal system. Each portal system runs link aggregation with a single aggregator. The distributed relay function enables the portal systems to jointly terminate a link aggregation group. To all other systems to which the portal is connected, the link aggregation group appears to terminate in a separate emulated system created by the portal systems.

Distributed Relay Control Protocol

In MC-LAG, the Distributed Relay Control Protocol (DRCP) perform these tasks:

- Establishes communication between portal systems across an Intra-Portal Link
- Verifies the consistent configuration of portal systems
- Determines the identity to use for the emulated system
- Distributes the current states of the portal systems and their aggregation ports among each other
- Computes the path of any frame passing through each IPL
- Exchanges information with adjacent portal systems to prevent forwarding loops and duplicate frame delivery

DRCP operation maintains the variables that control the forwarding of frames by the distributed relay function.

Supported Topologies

ZebOS-XP supports these types of MC-LAG topologies

- A portal with 1 portal system with no Intra Portal Link (IPL)
- A portal with 2 portal systems with a single IPL
- A portal with 3 portal systems with two IPLs
- A portal with 3 portal systems connected in a ring by 3 IPLs

ZebOS-XP Extension to MC-LAG

ZebOS-XP adds these features to MC-LAG:

- Master/slave election among portal systems in a portal
- Handling split-brain scenario
- Automatic allocation of gateway and port-conversation identifiers in a portal

Master/Slave Portal System Election

In a portal, member ports are distributed across the portal system and each portal system runs its own Layer 2 protocol stack. This can result in sending multiple control packets to the remote node. So, ZebOS-XP selects a master and slave among the portal systems in a portal and the master node takes charge of sending the control packets in the portal. Figure 10-4 shows how node 1 has been selected as the master in the previous configuration.

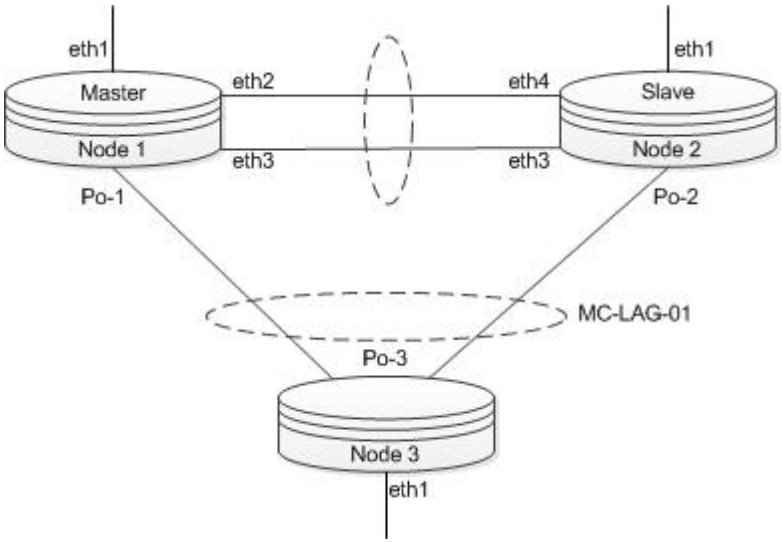

Figure 10-4: Master-slave selection

ZebOS-XP makes the following assumptions for master/slave election:

- Each system in a portal is configured with a unique number. If not, the election procedure does not complete and waits until a network administrator configures the system numbers for all system in the portal.
- As per the 802.1AX-REV/D3.1, ZebOS-XP supports a maximum of 3 portal systems in a portal whose subsystem numbers range from 1 to 3.

The master/slave election procedure is considered as part of the DRCP receive state machine. In the CURRENT state in the DRCP receive state machine, ZebOS-XP compares the portal system numbers with the other portal systems. The portal system with the lowest system number is the master, while the other portal systems are slave portal systems.

If the master portal system is down or not reachable by the other portal systems, then the remaining portal systems restart the election of master/slave portal systems resulting in the MC-LAG interface being down for all control plane protocols until the election process completes.

Handling Split-Brain Scenario

In a split-brain scenario, the IPL connecting portal systems is down. All the portal systems revert to the default system address used in LACP instead of their portal address. As a result, the LAG interface on the remote node is re-created with a new partner system address which results in traffic loss for about 3 seconds and uncertainty about which systems in the portal forms the LAG.

To avoid this non-deterministic behavior, when the IPL is down, ZebOS-XP sets the LACP system priority to the portal system number value in every portal system. The LACP system priority of the master portal system is lower than the other portal systems, which results in successfully forming a LAG with the remote node.

Automatic Allocation of Conversation Identifiers

As per 802.1AX-REV-D3.1, a network administrator must manually configure gateway and port conversation identifier mappings with portal systems and port identifiers. To avoid this manual configuration, a dynamic mechanism to distribute conversation identifiers among the portal systems in a portal is used.

The auto-allocation feature makes the following assumptions for a portal system:

- There is one-to-one mapping between VLAN identifiers and conversation identifiers.
- There is one-to-one mapping between gateway conversation identifiers and port conversation identifiers.
- All portal systems in a portal are configured for automatic conversation identifier allocation.

- To split the conversation identifiers evenly between the aggregation ports in a portal, the portal system supports a proprietary Additional Info TLV. In case of operational down status of any aggregation port, the conversation identifier will choose other portal systems aggregation port based on the priority.

The auto-allocation feature makes the following assumption for a remote node:

- For LACP conversation-sensitive frame collection and distribution to work properly in a remote node, LACP must adapt to the auto conversation identifier allocation performed by a portal.

ZebOS-XP also assumes that enabled VLANs are the same across the portal systems and the MD5 digest for enabled VLANs are shared between the portal system for verification. If all portal systems have same digest, then each portal system can split the conversation identifiers equally between them on its own. There is no dependency with the master portal system for conversation identifier calculation.

Gateway Conversation Identifier Calculation

Based on the number of portal systems in the portal topology and the number of VLANs, conversation identifiers are split. Since VLANs are the same in a portal, the VLANs can be divided equally among the portal systems:

- The first portal system takes the first set
- The second portal system takes the second set
- The third portal system takes the remaining with priority 1

The configured portal system number is used to choose the conversation identifier.

For example, a 3-portal system topology has 1000 VLANs configured in each portal system on the port-channel mapped to the current portal:

- Portal system 1 takes VLAN 1-333. The same conversation identifier is mapped with portal system 2 with priority 2 and with portal system 3 with priority 3 in round-robin fashion.
- Portal system 2 takes VLAN 334-666. The same conversation identifier is mapped with portal system 3 with priority 2 and with portal system 1 with priority 3 in round-robin fashion.
- Portal system 3 takes VLAN 667-1000. The same conversation identifier is mapped with portal system 1 with priority 2 and with portal system 2 with priority 3 in round-robin fashion.

The conversation identifier allocations for this example are shown in Table 10-1.

Table 10-1: Conversation identifier allocation for a 3-portal system with 1000 VLANs

Portal system number	Priority 1	Priority 2	Priority 3
1	1-333	667-1000	334-666
2	334-666	1-333	667-1000
3	667-1000	334-666	1-333

When 1000 VLANs are divided, each portal system gets 333.3 shares. Each portal system takes 333 and the last portal system takes the remainder. The default VLAN is handled as part of the normal conversation identifier split.

Recalculating a gateway conversation identifier is required when:

- VLAN addition/deletion: In this case, there will be mismatch in the VLAN digest, so the gateway and port conversation identifier state will go OUT-OF-SYNC with neighbor and the MC-LAG instance will be operationally down. Once the gateway and port conversation identifier state goes out-of-sync, the portal system does not use the portal address for the system identifier, so it will not be part of the portal.

  Recalculation is triggered once the VLAN digests are the same.

- Portal topology change: When there is change in portal topology (such as when the number of portal systems goes from 2 to 3 or vice versa), then the automatic gateway conversation identifier allocation is triggered.

When a portal system goes down or comes up, automatic allocation of gateway conversation identifier is not triggered. In this case, the portal topology (the topology state variable as set by the network administrator) is unchanged, but DRCP time-outs for the neighbor, but the data path still flows through the conversation identifier with priority 2.

Port Conversation Identifier Calculation

Once the gateway conversation identifier is done for a portal system, the computed conversation identifier is divided evenly between the aggregation ports. There is a one-to-one mapping between a gateway conversation identifier and a port conversation identifier.

In port conversation identifier calculation:

- Priorities 1 and 2 for the conversation identifier are chosen within the portal system
- Priorities 3 and 4 are used on the other portal system

If any one of the aggregation links is down, then the other link with priority 2 on same portal system takes charge of the conversation identifier (link-resilience). If the portal system is down, then the other portal system with priority 3 for the same conversation identifier takes charge (node resilience).

For example, assume there are 2 aggregation ports (`eth0` and `eth1`) on portal system 1 and conversation identifiers 1-333 are associated with it. The first aggregation port gets the first half and the second aggregation port gets the other half:

- Port `eth0` takes conversation identifiers 1-166 with priority 1
  - The same conversation identifiers are allocated to `eth1` of the same portal system with priority 2
  - Priority 3 and priority 4 are assigned to the other portal system
- Port `eth1` takes conversation identifiers 167-333 with priority 1
  - The same conversation identifiers are allocated to `eth0` of the same portal system with priority 2
  - Priority 3 and priority 4 will be assigned to other portal system

The conversation identifier allocations for this example are shown in Table 10-2.

Table 10-2: Conversation identifier allocation for 2 aggregation ports

Port	Priority 1	Priority 2	Priority 3	Priority 4
eth0	1-166	167-333	Assigned to other portal system	
eth1	167-333	1-166	Assigned to other portal system	

Port conversation identifier calculation is retriggered in case of:

- VLAN addition/deletion: The digest changes and both gateway and port automatic conversation recalculation is done once the VLAN digests become the same.
- Aggregation port addition/deletion: Once gateway conversation identifier allocation is done for the portal system, whatever an aggregation is port added or deleted, gateway conversation identifier (allocated to the portal system) is shared between the aggregation ports. If there is no aggregation port associated with the portal system, then the port conversation identifier with priority 2, which is associated with the other portal system, takes charge of the conversation identifier.
- Portal topology change: Automatic port conversation is triggered once the allocation of gateway conversation is done.

If the portal system goes up or down, then the port conversation identifier with the priority that is associated with other portal system takes charge.

Note: In a linear 3-node portal topology, if the middle node goes down due to mismatch in VLAN digest, then it results in split-brain scenario.

MC-LAG Daemon

Link aggregation is implemented in ZebOS-XP as part of the `mlagd` daemon which manages LACP and DRNI as sub-modules as shown in Figure 10-5.

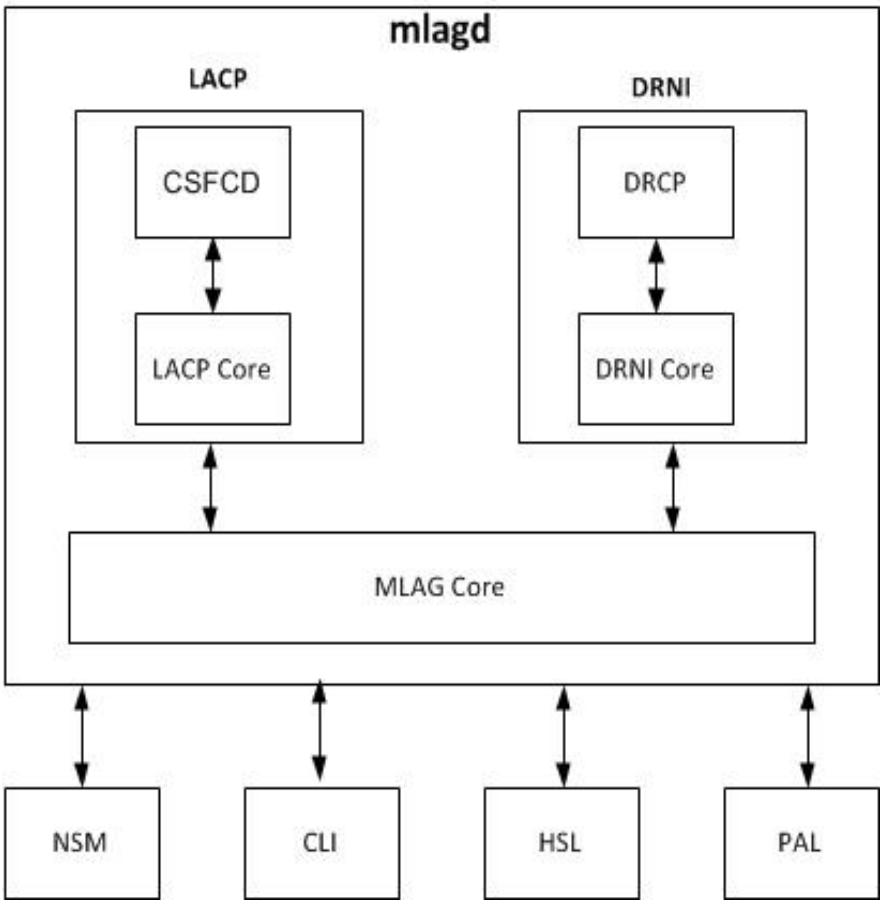

Figure 10-5: mlagd architecture

In the LACP module:

- The LACP core sub-module implements LACP functionality per the IEEE-802.1AX-2008 and IEEE-802.1AX-REV/D3.1 standards.
- The Conversation Sensitive Frame Collection and Distribution (CSFCD) sub-module implements these mechanisms:
  - The Frame Collector only accepts frames received on an expected aggregation link
  - The Frame Distributor selects the aggregation link for any given frame

  Instead of using a traditional hash-based mechanism to distribute traffic between aggregation ports in a port-channel, the CSFCD module distributes traffic based on conversation identifier mapping among aggregation ports per the IEEE-802.1AX-REV/D3.1 standard.

In the DRNI module:

- The DRNI core sub-module is responsible for configuration management, automatic allocation of gateway/port conversation identifiers, and master-slave election between the portal systems.
- The DRCP sub-module implements state machines and exchanges DRCPDUs (Distributed Relay Control Protocol Data Units) with neighbor portal systems forming a portal.

The MLAG core module interacts with NSM, CLI, HSL, and PAL.

API

Data Structures

This section describes the common data structures that are used with link aggregation.

The functions in this chapter refer to these data structures which are described in the *Common Data Structures Developer Guide*:

- `cli`
- `interface`
- `nsm_master`

lacp_link

This data structure represents a link aggregation group is defined in `mlagd/lacp/lacp_types.h`:

Member	Definition
`next`	Next link in list
`aggregator`	Aggregator (parent)
`ifp`	Interface
`config_channel_id`	Configuration channel identifier
`agg_link_index`	Index of link in the aggregator
`mac_addr`	Local MAC address of the link
`name`	Administrative name
`lacp_pdu`	Decoded PDU - Used in place of CtrlMuxN:MA_DATA if not null
`rcv_state`	Receive state
`periodic_tx_state`	Periodic transmitting state
`mux_machine_state`	Mux machine state
`actor_churn_state`	Actor churn state
`partner_churn_state`	Partner churn state
`actor_churn_count`	Number of times actor churn state machine entered an ACTOR_CHURN state
`partner_churn_count`	Number of times partner churn state machine entered PARTNER_CHURN state
`actor_sync_transition_count`	Number of times actor MUX state entered IN_SYNC state

Member	Definition
partner_sync_transition_count	Number of times partner MUX state entered IN_SYNC state
current_while_timer	Current while timer
actor_churn_timer	Actor churn timer
periodic_timer	Periodic timer
partner_churn_timer	Partner churn timer
wait_while_timer	Wait while timer)
current_while_timer_expired	Timer expiry flag
actor_churn_timer_expired	Timer expiry flag
periodic_timer_expired	Timer expiry flag
partner_churn_timer_expired	Timer expiry flag
ntt	Need To Transmit flag: new protocol information that should be transmitted on the link or the partner needs to be reminded of the old information
port_enabled	The physical layer has indicated that the link has been established and the port is operable
lacp_enabled	The port is operating the LACP
actor_churn	The Actor Churn Detection machine has detected that local port configuration has failed to converge within a specified time and management intervention is required
partner_churn	The Partner Churn Detection machine has detected that remote port configuration has failed to converge within a specified time and management intervention is required
ready_n	The wait_while_timer has expired and it is waiting (the port is in the WAITING state) to attach to an Aggregator
selected	Whether the Selection Logic has selected an appropriate Aggregator
port_moved	The Receive machine for a port is in the PORT_DISABLED state and the combination of Partner_Oper_System and Partner_Oper_Port_Number in use by that port has been received in an incoming LACPDU on a different port
actor_port_number	Actor port number; system-wide unique index of port
actor_port_priority	Actor port priority
actor_port_aggregator_identifier	Identifier of the Aggregator to which this port is attached
actor_admin_port_key	Actor administrative port key
actor_oper_port_key	Actor operational port key

Member	Definition
actor_admin_port_state	Administrative port state
actor_oper_port_state	Actor operational port state
partner_admin_system	Partner administrative
partner_oper_system	Partner operational system
partner_admin_system_priority	Partner administrative system priority
partner_oper_system_priority	Partner operational priority
partner_admin_key	Partner administrative key
partner_oper_key	Partner operational key
partner_admin_port_number	Partner administrative port number
partner_oper_port_number	Partner operational port number
partner_admin_port_priority	Partner administrative port priority
partner_oper_port_priority	Partner operational port priority
partner_admin_port_state	Partner administrative port state
partner_oper_port_state	Partner operational port state
partner_change_count	Number of times the partner's perception of the LAG ID for this aggregation port has changed
tx_count	Transmission count
lacpdu_sent_count	Number of LACPDUs sent
lacpdu_recv_count	Number of LACPDUs received
mpdu_recv_count	MACsec Key Agreement Protocol Data Units (MPDUs) received
mpdu_sent_count	MPDUs sent
pckt_sent_err_count	Packets sent error count
pckt_recv_err_count	Packets received error count
mpdu_response_recv_count	MPDUs responses received
mpdu_response_sent_count	MPDUs responses sent
pckt_unknown_rx_count	Packets unknown received
last_pdu_rx_time	Time last PDU was received
received_marker_info	Received marker information
agg_link_index	Index of link in the aggregator

Member	Definition
flags	Flags: LACP_HA_LINK_STALE_FLAG LINK_FLAG_AGG_MATCH
lacp_link_cdr_ref	CDR reference for lacp_link
current_while_timer_cdr_ref	CDR reference for current_while timer
actor_churn_timer_cdr_ref	CDR reference for actor_churn_timer
periodic_timer_cdr_ref	CDR reference for periodic timer
partner_churn_timer_cdr_ref	CDR reference for partner churn timer
wait_while_timer_cdr_ref	CDR reference for wait while timer
attach_state	Attach state
protocol_da	Protocol destination address
partner_lacpdu_version_number	Partner LACPDU version number
csfcdl	Conversation Sensitive Frame Collection and Distribution (CSFCD) sub-module reference
enable_long_pdu_xmit	Enable long PDU transmit

Definition

```
struct lacp_link
{
  /* Housekeeping */
  struct lacp_link        * next;       /* List of all links. */
  struct lacp_aggregator  * aggregator;

  struct interface *ifp;

  u_int32_t config_channel_id;

  /* Index of link in the aggregator */
  s_int32_t agg_link_index;

  /* Local mac address of the link */
  u_char             mac_addr[LACP_GRP_ADDR_LEN];
  u_char             name[LACP_IFNAMSIZ];  /* Administrative name */

  struct lacp_pdu *         pdu;      /* Decoded PDU - Used in place
                                         of CtrlMuxN:MA_DATA if not null. */
  /* State tracking */
  enum lacp_rcv_state             rcv_state;
  enum lacp_periodic_tx_state     periodic_tx_state;
  enum lacp_mux_state             mux_machine_state;
```

```
enum lacp_actor_churn_state     actor_churn_state;
enum lacp_partner_churn_state   partner_churn_state;

/* Timers */
struct thread *current_while_timer;
struct thread *actor_churn_timer;
struct thread *periodic_timer;
struct thread *partner_churn_timer;
struct thread *wait_while_timer;

/* Timer expiry flags */
u_char  current_while_timer_expired:1;
u_char  actor_churn_timer_expired:1;
u_char  periodic_timer_expired:1;
u_char  partner_churn_timer_expired:1;

/* Link variables */
unsigned int  ntt:1;
unsigned int  port_enabled:1;
unsigned int  lacp_enabled:1;     /* 43.4.8 */
unsigned int  actor_churn:1;      /* 43.4.8 */
unsigned int  partner_churn:1;    /* 43.4.8 */
unsigned int  ready_n:1;          /* 43.4.8 */
unsigned int  selected:2;         /* 43.4.8 */
unsigned int  port_moved:1;       /* 43.4.8 */

u_int8_t    actor_system[LACP_GRP_ADDR_LEN];
u_int16_t   actor_system_priority;

u_int16_t   actor_port_number;      /* System-wide unique index of port */
u_int16_t   actor_port_priority;
u_int32_t   actor_port_aggregator_identifier;
u_int16_t   actor_admin_port_key;
u_int16_t   actor_oper_port_key;
u_int8_t    actor_admin_port_state;
u_int8_t    actor_oper_port_state;

u_int8_t    partner_admin_system[LACP_GRP_ADDR_LEN];
u_int8_t    partner_oper_system[LACP_GRP_ADDR_LEN];
u_int16_t   partner_admin_system_priority;
u_int16_t   partner_oper_system_priority;
u_int16_t   partner_admin_key;
u_int16_t   partner_oper_key;
u_int16_t   partner_admin_port_number;
u_int16_t   partner_oper_port_number;
u_int16_t   partner_admin_port_priority;
u_int16_t   partner_oper_port_priority;
u_int8_t    partner_admin_port_state;
u_int8_t    partner_oper_port_state;
```

```
  /* number of times the Partner's perception of
   * the LAG ID (see 43.3.6.1) for this Aggregation Port has changed*/
  u_int16_t partner_change_count;

  /* Number of times Actor churn state machine
   * has entered ACTOR_CHURN state */
  u_int32_t         actor_churn_count;

  /* Number of times Partner churn state machine
   * has entered PARTNER_CHURN state */
  u_int32_t         partner_churn_count;

  /* Number of times Actor Mux state
   * has entered IN_SYNC state */
  u_int32_t         actor_sync_transition_count;

  /* Number of times Partner Mux state
   * has entered IN_SYNC state */
  u_int32_t         partner_sync_transition_count;

  /* Used to track LACPDU transmissions */
  u_int16_t  tx_count;
  /*Used to count no of LACPDUs sent or recieved*/
  u_int32_t lacpdu_sent_count;
  u_int32_t lacpdu_recv_count;
  u_int32_t mpdu_recv_count;
  u_int32_t mpdu_sent_count;
  u_int32_t pckt_sent_err_count;
  u_int32_t pckt_recv_err_count;
  u_int32_t mpdu_response_recv_count;
  u_int32_t mpdu_response_sent_count;
  u_int32_t pckt_unknown_rx_count;

  /* Time when the last pdu has been received */
  pal_time_t last_pdu_rx_time;
  struct marker_pdu received_marker_info;

#define LINK_FLAG_AGG_MATCH      (1 << 0)
#define LACP_HA_LINK_STALE_FLAG  (1 << 1)
  u_char flags;

#ifdef HAVE_HA
  HA_CDR_REF lacp_link_cdr_ref; /* CDR ref for lacp_link */
  HA_CDR_REF current_while_timer_cdr_ref; /* CDR ref for current_while timer */
  HA_CDR_REF actor_churn_timer_cdr_ref; /* CDR ref for actor_churn_timer */
  HA_CDR_REF periodic_timer_cdr_ref; /* CDR ref for periodic timer */
  HA_CDR_REF partner_churn_timer_cdr_ref; /* CDR ref for partner churn timer */
  HA_CDR_REF wait_while_timer_cdr_ref; /* CDR ref for wait while timer */
#endif /* HAVE_HA */
  bool_t attach_state;
```



```
  u_int8_t protocol_da[6];

#ifdef HAVE_LACPV2
  u_int32_t partner_lacpdu_version_number;
  struct lacp_link_csfcd *csfcdl;

  bool_t enable_long_pdu_xmit;
#endif /*HAVE_LACPV2*/
};
```

Functions

The functions in this chapter are used with link aggregation.

Function	Description
drni_api_dbg_show_ipp_details	Displays information about an intra-portal port
drni_api_dbg_show_portal_detail	Displays information about MC-LAG
drni_api_mac_address_is_valid	Validates a MAC address
drni_api_set_conv_alloc_mode	Sets the conversation identifier allocation mode
drni_api_set_intra_portal_link	Creates an intra-portal link
drni_api_set_ipp_periodic_time	Sets the timeout for DRCPDU exchange
drni_api_set_portal_address	Sets the MAC address of a portal system
drni_api_set_portal_name	Sets the name of a portal
drni_api_set_portal_priority	Sets the portal priority of a portal system
drni_api_set_portal_system_number	Sets the number of a system in a portal
drni_api_set_portal_topology	Sets the topology of a portal system
drni_api_unset_intra_portal_link	Removes an intra-portal link
drni_api_update_gateway_conv	Allocates or removes a range of gateway conversation identifiers
drni_clear_statistics_all	Clears the DRCPDU statistics for the portal system
drni_set_destination_address_type	Sets the address type to use for sending DRCPDUs on an IPP
drni_show_gateway_conv_detail	Displays MC-LAG gateway conversation identifiers
drni_show_mlag_summary	Displays MC-LAG configuration and status.
drni_show_portal_detail	Displays MC-LAG portal conversation identifiers
drni_show_portal_summary	Displays portal configuration and status
drni_show_port_conv_detail	Displays MC-LAG portal conversation identifiers
drni_show_statistics_all	Displays DRCPDU statistics

Function	Description
lacp_api_cli_show_debug_po_details	Displays collection and port conversation masks
lacp_api_cli_show_etherchannel_port_conv	Displays port conversation identifier allocation
lacp_api_discard_wrong_conversation	Enables or disables discarding frames with an incorrect port conversation identifier
lacp_api_set_aggregation_port_destination_addr_type	Sets the address type to use for sending LACPDUs
lacp_find_link_by_name	Search for a link under LACP control
lacp_set_channel_priority	Sets a priority level for an LACP channel
lacp_set_channel_timeout	Sets the LACP timeout on a per-port basis
lacp_set_system_priority	Sets a system priority level of the LACP channel
lacp_unset_channel_priority	Unsets the LACP channel priority for port
lacp_unset_system_priority	Unsets the system priority level of the LACP channel
mlag_api_set_mlag_instance	Creates an MC-LAG instance
mlag_api_unset_mlag_instance	Deletes an MC-LAG instance
mlag_api_update_port_conv	Allocates or removes a range of port conversation identifiers
nsm_lacp_aggregator_psc_set	Sets the load-balancing mode for an aggregator
nsm_lacp_api_add_aggregator_member	Adds an aggregator member
nsm_lacp_api_delete_aggregator_member	Deletes an aggregator member
nsm_mlag_api_add_aggregator_member	Assigns an MLAG identifier to an interface (port channel)
nsm_mlag_api_delete_aggregator_member	Removes an MLAG identifier from a port channel

drni_api_dbg_show_ipp_details

This function displays information about an intra-portal port.

This function is called by this command:

```
show dbg mlag <1-65535> ipp
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_api_dbg_show_ipp_details (struct cli *cli, u_int16_t portal_id)
```

Input Parameters

`cli`	CLI structure
`portal_id`	MLAG identifier

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal for `portal_id` cannot be found

drni_api_dbg_show_portal_detail

This function displays information about MC-LAG.

This function is called by this command:

```
show dbg mlag <1-65535>
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_api_dbg_show_portal_detail (struct cli *cli, u_int16_t portal_id)
```

Input Parameters

`cli`	CLI structure
`portal_id`	MLAG identifier

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal for `portal_id` cannot be found

drni_api_mac_address_is_valid

This function validates a MAC address, ensuring that it is in `HHHH.HHHH.HHHH` format.

This function is called by this command:

```
portal-address
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_mac_address_is_valid (u_char *mac_addr)
```

Input Parameters

`mac_addr`	MAC address

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_FAILURE when the function fails

drni_api_set_conv_alloc_mode

This function sets the conversation identifier allocation mode.

This function is called by this command:

```
conversation alloc-mode
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_conv_alloc_mode (u_int16_t drni_id, u_int8_t mode)
```

Input Parameters

`drni_id` MC-LAG identifier

`mode` Allocation mode; one of these constants from `mlagd/drni/drni_api.h`:

`DRNI_AUTO_CONV_MODE`
Automatic conversation allocation

`DRNI_MANUAL_CONV_MODE`
Manual conversation allocation

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_PROP_INFO_NULL if the home property information is NULL

drni_api_set_intra_portal_link

This function creates an intra-portal link.

This function is called by this command:

```
intra-portal-link
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_intra_portal_link (u_int16_t drni_id, char *ifname,
                                u_int8_t neigh_ps_no)
```

Input Parameters

`drni_id`	MC-LAG identifier
`ifname`	Interface name to set as an intra-portal port (IPP)
`neigh_ps_no`	Neighbor portal system number

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_NULL_POINTER when `ifname` is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_PORTAL_SYS_NUM when `neigh_ps_no` is out of range <1-3>

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_IFP_NOT_FOUND when the interface is not found

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_PORTAL_SYS_NUM_EXIST when `neigh_ps_no` is already used

DRNI_ERR_SAME_NEIGH_EXIST when some other link has been configured as an IPP for `neigh_ps_no`

DRNI_ERR_MAX_IPP_REACHED when the maximum number of IPPs are already configured

DRNI_ERR_CALLOC_FAILED when memory allocation fails

drni_api_set_ipp_periodic_time

This function sets the timeout for DRCPDU exchange.

This function is called by this command:

```
intra-portal-link
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_ipp_periodic_time (u_int16_t drni_id, char *ifname,
                                u_int8_t mode)
```

Input Parameters

`drni_id`	MC-LAG identifier
`ifname`	Interface name used as an intra-portal port (IPP)
`mode`	Timeout for DRCPDU exchange; one of these constants from `mlagd/drni/drni_api.h`:

`DRNI_API_SHORT_TIMEOUT`

One second between periodic transmissions

`DRNI_API_LONG_TIMEOUT`

Thirty seconds between periodic transmissions

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_IFP_NOT_FOUND when the interface is not found

DRNI_FAILURE when `mode` is invalid

DRNI_ERR_IPP_NOT_FOUND when the IPP cannot be found for the interface

drni_api_set_portal_address

This function sets the MAC address of a portal system.

This function is called by this command:

```
portal-address
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_portal_address (u_int16_t drni_id, u_char *mac)
```

Input Parameters

`drni_id`	MC-LAG identifier
`mac`	MAC address

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_NULL_POINTER when `mac` is NULL

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_PORTAL_ADDR when `mac` is zero or a broadcast MAC (`000.0000.0000` and `FFFF.FFFF.FFFF` respectively)

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

drni_api_set_portal_name

This function sets the name of the portal.

This function is called by this command:

```
portal-name
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_portal_name (u_int16_t drni_id, char *portal_name)
```

Input Parameters

`drni_id`	MC-LAG identifier
`portal_name`	Portal name

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_NULL_POINTER when `portal_name` is NULL

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

drni_api_set_portal_priority

This function sets the portal priority.

This function is called by this command:

```
portal-priority
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_portal_priority (u_int16_t drni_id, u_int16_t portal_priority)
```

Input Parameters

`drni_id`	MC-LAG identifier
`portal_priority`	Portal priority

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_PORTAL_PRIORITY when `portal_priority` is out of range <0-65535>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

drni_api_set_portal_system_number

This function sets the number of a system in a portal.

This function is called by this command:

```
portal-system-number
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_portal_system_number (u_int16_t drni_id, u_int8_t ps_num)
```

Input Parameters

`drni_id`	MC-LAG identifier
`ps_num`	Portal system number

Output Parameters

None

Return Value

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_PORTAL_SYS_NUM when `ps_num` is out of range <1-3>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_PORTAL_SYS_NUM_EXIST when `ps_num` is already used

drni_api_set_portal_topology

This function sets the topology of a portal system.

This function is called by this command:

```
portal-topology
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_set_portal_topology (u_int16_t drni_id, u_int8_t portal_topo)
```


Input Parameters

`drni_id` MC-LAG identifier

`portal_topo` Address type; one of these constants form `mlagd/drni/drni_api.h`:

`DRNI_API_1NODE_TOPOLOGY`
One-node topology

`DRNI_API_2NODE_TOPOLOGY`
Two-node topology

`DRNI_API_3NODE_TOPOLOGY`
Three-node topology

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_TOPOLOGY when `portal_topo` is not valid

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_NULL_POINTER when the home property information is NULL

drni_api_unset_intra_portal_link

This function removes an intra-portal link.

This function is called by this command:

```
no intra-portal-link
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_unset_intra_portal_link (u_int16_t drni_id, char * ifname)
```

Input Parameters

`drni_id` MC-LAG identifier

`ifname` Interface name used as an intra-portal port (IPP)

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_NULL_POINTER when `ifname` is NULL

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_IFP_NOT_FOUND when the interface is not found

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_IPP_NOT_FOUND when the IPP cannot be found for the interface

drni_api_update_gateway_conv

This function manually allocates or removes a range of gateway conversation identifiers. This range of gateway conversation identifiers is distributed through the configured portal system.

This function is called by this command:

```
gateway-conv-id <1-4096> (<1-4096>|) [portal-system <1-3> priority <1-3>]
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_api_update_gateway_conv (u_int16_t drni_id, int count,
                              char **config, u_int8_t flag)
```

Input Parameters

`drni_id`	MC-LAG identifier
`count`	Number of command-line parameters in `config`
`config`	Command-line parameters
`flag`	Whether adding or removing a gateway conversation identifier allocation; one of these constants defined in `mlagd/mlag_api.h`:

`MLAG_API_ADD_CONV_CONFIG`

Add gateway conversation identifier allocation

`MLAG_API_DEL_CONV_CONFIG`

Remove gateway conversation identifier allocation

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_FAILURE when `config` is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_INVALID_CONV_ID when the starting or ending conversation identifier is invalid

DRNI_ERR_CVID_UNSUPPORTED_CONV_ID when the starting or ending conversation identifier is greater than 4094

DRNI_ERR_INVALID_CONV_ID when the starting and ending conversation identifier are out of range <1-4096>

DRNI_ERR_INVALID_PORTAL_SYS_NUM when the portal system number is out of range <1-3>

DRNI_ERR_INVALID_PORTAL_PRIORITY when the portal priority is out of range <1-3>

DRNI_ERR_INVALID_PRIORITY when the priority is out of range <1-4>

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_AUTO_ALLOC_SET when the portal is set for automatic conversation identifier allocation

DRNI_ERR_CONFIG_OVERLAPS when a given conversation identifier is not configured for the portal system number or the priority

DRNI_ERR_CALLOC_FAILED when memory allocation fails

drni_clear_statistics_all

This function clears the DRCPDU (Distributed Relay Control Protocol Data Unit) statistics for the portal system.

This function is called by this command:

```
clear drcpdu-statistics
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_clear_statistics_all (struct cli *cli)
```

Input Parameters

`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

drni_set_destination_address_type

This function sets the address type to use for sending DRCPDUs (Distributed Relay Control Protocol Data Units) on an IPP (Intra-Portal Port).

This function is called by this command:

```
intra-portal destination-address-type
```

Syntax

```
#include "mlagd/drni/drni_api.h"
s_int32_t
drni_set_destination_address_type (u_int32_t drni_id, u_int8_t addr_type)
```

Input Parameters

`drni_id`	MC-LAG identifier
`addr_type`	Address type; one of these constants from `mlagd/drni/drni_api.h`:

`DRNI_NR_CUST_BR_ADDR`

Customer bridge group address

`DRNI_NR_BR_ADDR`

Multicast group address

`DRNI_NR_NON_TPMR_BR_ADDR`

Non-Two-Port Media Access Control Relay (TPMR) group address (default)

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_INVALID_DRNI_ID when `drni_id` is out of range <1-65535>

DRNI_ERR_NULL_POINTER when `portal_name` is NULL

DRNI_ERR_PORTAL_NULL when the portal for `drni_id` cannot be found

DRNI_ERR_INVALID_ADDR_TYPE when `addr_type` is not valid

drni_show_gateway_conv_detail

This function displays MC-LAG gateway conversation identifiers.

This function is called by this command:

```
show mlag <1-65535> gateway-conversation-id
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_gateway_conv_detail (u_int16_t portal_id, struct cli *cli)
```

Input Parameters

`portal_id`	MLAG identifier
`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_PORTAL_NULL when the portal cannot be found

drni_show_mlag_summary

This function displays MC-LAG configuration and status.

This function is called by this command:

```
show mlag (<1-65535>|) summary
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_mlag_summary (struct cli *cli)
```

Input Parameters

`portal_id`	MLAG identifier
`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal cannot be found

drni_show_portal_detail

This function displays details about MC-LAG configuration and status.

This function is called by this command:

```
show mlag <1-65535> detail
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_portal_detail (u_int16_t portal_id, struct cli *cli)
```

Input Parameters

`portal_id`	MLAG identifier
`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal cannot be found

drni_show_portal_summary

This function displays portal configuration and status.

This function is called by this command:

```
show mlag (<1-65535>|) summary
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_portal_summary (u_int16_t portal_id, struct cli *cli)
```

Input Parameters

`portal_id`	MLAG identifier
`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal cannot be found

drni_show_port_conv_detail

This function displays MC-LAG portal conversation identifiers.

This function is called by this command:

```
show mlag <1-65535> port-conversation-id
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_port_conv_detail (u_int16_t portal_id, struct cli *cli)
```

Input Parameters

`portal_id`	MLAG identifier
`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_PORTAL_NULL when the portal cannot be found

drni_show_statistics_all

This function displays DRCPDU statistics.

This function is called by this command:


```
show drcpdu statistics
```

Syntax

```
#include "mlagd/drni/drni_show.h"
s_int32_t
drni_show_statistics_all (struct cli *cli)
```

Input Parameters

`cli`	CLI structure

Output Parameters

None

Return Values

DRNI_SUCCESS when the function succeeds

DRNI_ERR_MASTER_NULL when the DRNI master is NULL

DRNI_ERR_PORTAL_NULL when the portal cannot be found

DRNI_FAILURE when the function fails

lacp_api_cli_show_debug_po_details

This function displays collection and port conversation masks.

This function is called by this command:

```
show debug po
```

Syntax

```
#include "mlagd/lacp/lacp_show.h"
void
lacp_api_cli_show_debug_po_details (struct cli *cli, u_int32_t key)
```

Input Parameters

`cli`	CLI structure
`key`	MLAG identifier

Output Parameters

None

Return Values

None

lacp_api_cli_show_etherchannel_port_conv

This function displays port conversation identifier allocation.

This function is called by this command:

```
show etherchannel <1-65535> port-conversation-id
```

Syntax

```
#include "mlagd/lacp/lacp_show.h"
void
lacp_api_cli_show_etherchannel_port_conv (struct cli *cli, u_int32_t key)
```

Input Parameters

`cli`	CLI structure
`key`	MLAG identifier

Output Parameters

None

lacp_api_discard_wrong_conversation

This function enables or disables discarding frames with an incorrect port conversation identifier.

This function is called by this command:

```
lacp discard wrong conversation
```

Syntax

```
#include "mlagd/lacp/lacp_api.h"
s_int32_t
lacp_api_discard_wrong_conversation (u_int8_t *agg_name, bool_t discard)
```

Input Parameters

`agg_name`	Aggregator name
`discard`	Whether to discard frames; one of these constants from `pal/dummy/pal_types.h`:
`PAL_TRUE`	Discard frames with an incorrect port conversation identifier
`PAL_FALSE`	Do not discard frames with an incorrect port conversation identifier

Output Parameters

None

Return Values

LACP_SUCCESS when the function succeeds

LACP_FAILURE when `agg_name` is NULL

LACP_ERR_AGG_NOT_FOUND when the aggregator cannot be found

LACP_ERR_AGG_CSFCD_NOT_FOUND when the aggregator does not have a reference to the CSFCD (Conversation Sensitive Frame Collection and Distribution) module

lacp_api_set_aggregation_port_destination_addr_type

This function sets the address type to use for sending Link Aggregation Control Protocol Data Units (LACPDUs).

This function is called by this command:

```
lacp destination-mac
```


Syntax

```
#include "mlagd/lacp/lacp_api.h"
s_int32_t
lacp_api_set_aggregation_port_destination_addr_type (u_int8_t *if_name,
                                                     u_int8_t type)
```

Input Parameters

`if_name` Interface name

`type` Address type; one of these constants from `mlagd/lacp/lacp_config.h`:

`LACP_NR_CUST_BR_ADDR`

Customer bridge group address (default)

`LACP_MCAST_GRP_ADDR`

Multicast group address

`LACP_NR_NON_TPMR_BR_ADDR`

Non-Two-Port Media Access Control Relay (TPMR) group address

Output Parameters

None

Return Values

LACP_SUCCESS when the function succeeds

LACP_FAILURE when `if_name` is NULL or `type` is invalid

LACP_ERR_LINK_NOT_FOUND when the link cannot be found

lacp_find_link_by_name

This function searches for a link aggregation group.

This function is called by several different LACP commands.

Syntax

```
#include "mlagd/lacp/lacp_link.h"
struct lacp_link *
lacp_find_link_by_name (const char *const name)
```

Input Parameters

`name` Link aggregation group name

Output Parameters

None

Return Value

Pointer to the link aggregation group when the function succeeds

`NULL` when the function fails

lacp_set_channel_priority

This function sets the priority level for a link aggregation group.

This function is called by this command:

```
lacp port-priority
```

Syntax

```
#include "lacpd/lacpd.h"
int
lacp_set_channel_priority (struct lacp_link *link, unsigned int priority);
```

Input Parameters

`link`	Link aggregation group
`priority`	Priority <1-65535>

Output Parameters

None

Return Values

RESULT_OK when function succeeds

RESULT_ERROR when `priority` is out of range <1-65535>

lacp_set_channel_timeout

This function sets the LACP timeout for a link aggregation group.

This function is called by this command:

```
lacp timeout
```

Syntax

```
#include "lacpd/lacpd.h"
int
lacp_set_channel_timeout (struct lacp_link *link, int timeout)
```

Input Parameters

`link`	Link aggregation group
`timeout`	Timeout duration; one of these constants from `lacpd/lacp_types.h`:
	`LACP_TIMEOUT_SHORT`
	Short timeout
	`LACP_TIMEOUT_LONG`
	Long timeout

Output Parameters

None

Return Values

RESULT_OK when function succeeds

LACP_ERR_INVALID_LACP_TIMEOUT when `timeout` is not valid

RESULT_ERROR when `link` is NULL

lacp_set_system_priority

This function sets the LACP system priority level.

This function is called by this command:

```
lacp system-priority
```

Syntax

```
#include "lacpd/lacpd.h"
int
lacp_set_system_priority (unsigned int priority)
```

Input Parameters

`priority`	System priority <1-65535>

Return Values

RESULT_OK when function succeeds.

LACP_SET_ERR_INVALID_VAL when `priority` is out of range <1-65535>

lacp_unset_channel_priority

This function sets the priority for a link aggregation group to its default value (32768).

This function is called by this command:

```
no lacp port-priority
```

Syntax

```
#include "lacpd/lacpd.h"
int
lacp_unset_channel_priority (struct lacp_link *link)
```

Input Parameters

`link`	Link aggregation group

Output Parameters

None

Return Values

RESULT_OK when the function succeeds

RESULT_ERROR when `link` is NULL

lacp_unset_system_priority

This function sets the system priority to its default value (32768).

This function is called by this command:

```
no lacp system-priority
```

Syntax

```
#include "lacpd/lacpd.h"
int
lacp_unset_system_priority ()
```

Input Parameters

None

Output Parameters

None

Return Values

Always RESULT_OK

mlag_api_set_mlag_instance

This function creates an MC-LAG instance.

This function is called by this command:

```
mlag configuration
```

Syntax

```
#include "mlagd/mlag_api.h"
s_int32_t
mlag_api_set_mlag_instance (u_int32_t mlag_id)
```

Input Parameters

`mlag_id`	MLAG identifier

Output Parameters

None

Return Values

MLAG_SUCCESS when the function succeeds

MLAG_FAILURE when `mlag_id` is out of range <1-65535>

MLAG_ERR_CALLOC_FAILED when memory allocation fails

mlag_api_unset_mlag_instance

This function deletes an MC-LAG instance.

This function is called by this command:

```
no mlag configuration
```

Syntax

```
#include "mlagd/mlag_api.h"
s_int32_t
mlag_api_unset_mlag_instance (u_int32_t mlag_id)
```

Input Parameters

`mlag_id`	MLAG identifier

Output Parameters

None

Return Values

MLAG_SUCCESS when the function succeeds

MLAG_FAILURE when `mlag_id` is out of range <1-65535>

mlag_api_update_port_conv

This function manually allocates or removes a range of port conversation identifiers. This range of conversation identifiers is distributed through the configured portal system.

This function is called by this command:

```
port-conv-id <1-4096> (<1-4096>|) [port-priority <1-65535> port-number <1-65535>
portal-system <0-3> priority <1-4>]
```

Syntax

```
#include "mlagd/mlag_api.h"
s_int32_t
mlag_api_update_port_conv (u_int8_t *agg_name, int count,
                           char **config, u_int8_t flag)
```

Input Parameters

`agg_name`	Aggregator name
`count`	Number of command-line parameters in `config`
`config`	Command-line parameters
`flag`	Whether adding or removing a gateway conversation identifier allocation; one of these constants defined in `mlagd/mlag_api.h`:

`MLAG_API_ADD_CONV_CONFIG`

Add port conversation identifier allocation

`MLAG_API_DEL_CONV_CONFIG`

Remove port conversation identifier allocation

Output Parameters

None

Return Values

MLAG_SUCCESS when the function succeeds

MLAG_FAILURE when `agg_name` or `config` is NULL

MLAG_FAILURE when the starting or ending conversation identifier is invalid

MLAG_ERR_INVALID_CONV_ID when the starting conversation identifier is out of range <1-4094>

MLAG_ERR_CVID_UNSUPPORTED_CONV_ID when the starting or ending conversation identifier is greater than 4094

MLAG_FAILURE when the port priority, port number, portal system number, or portal priority is invalid

LACP_FAILURE when the port priority is out of range <1-65535>

LACP_ERR_INVALID_CONV_ID when the starting and ending conversation identifier are out of range <1-4094>

LACP_ERR_AGG_NOT_FOUND when the aggregator cannot be found

MLAG_ERR_CONFIG_OVERLAPS when a given conversation identifier is not configured for the port or priority

LACP_ERR_LINK_NOT_FOUND when the port is not mapped to aggregator

LACP_ERR_LINK_PRIORITY_MISMATCH when the port is not mapped to aggregator

LACP_ERR_WRONG_PORT_AGG_MATCH when the port is not mapped to aggregator

MLAG_ERR_CALLOC_FAILED when memory allocation fails

nsm_lacp_aggregator_psc_set

This function sets the load-balancing mode for an aggregator.

This function is called by this command:

```
port-channel load-balance
```

Syntax

```
#include "nsm/lacp/nsm_lacp.h"
int
nsm_lacp_aggregator_psc_set (struct interface *ifp, int psc)
```

Input Parameters

`ifp` Pointer to the interface.

`psc` Port selection criteria (source/destination and MAC/IP/port); one of these constants from `hal/L2/hal_l2.h`:

`HAL_LACP_PSC_DST_MAC`
Destination MAC address

`HAL_LACP_PSC_SRC_MAC`
Source MAC address

`HAL_LACP_PSC_SRC_DST_MAC`
Source and Destination MAC address

`HAL_LACP_PSC_SRC_IP`
Source IP address

`HAL_LACP_PSC_DST_IP`

Destination IP address

`HAL_LACP_PSC_SRC_DST_IP`

Source and Destination IP address

`HAL_LACP_PSC_SRC_PORT`

Source port address

`HAL_LACP_PSC_DST_PORT`

Destination TCP/UDP address

`HAL_LACP_PSC_SRC_DST_PORT`

Source and Destination TCP/UDP address

Return Values

NSM_ERR_INVALID_ARGS when `ifp` is NULL or contains an invalid member

NSM_ERR_INTERNAL when there is an internal error

NSM_SUCCESS when the function succeeds

nsm_lacp_api_add_aggregator_member

This function creates a link aggregation group.

This function is called by this command:

```
channel-group
```

Syntax

```
#include "nsm/lacp/nsm_lacp.h"
int
nsm_lacp_api_add_aggregator_member (struct nsm_master *nm,
                                    struct interface *mem_ifp, u_int32_t key,
                                    char activate, bool_t notify_lacp,
                                    u_char agg_type, bool_t add_port)
```

Input Parameters

`nm`	Pointer to NSM master.
`mem_ifp`	Pointer to the interface.
`key`	Link aggregation group; this function adds a prefix to this value:
	“po” for a dynamic link aggregation group
	“sa” for a static link aggregation group
`activate`	Passive or active mode:
`PAL_TRUE`	Enable LACP negotiation on the port.
`PAL_FALSE`	Disable LACP negotiation on the port.
`notify_lacp`	Whether to notify LACP.
`agg_type`	Aggregation type: one of these constants in `lib/nsm_message.h`:
`AGG_CONFIG_STATIC`	Static aggregation.
`AGG_CONFIG_LACP`	

	Dynamic aggregation.
add_port	Set this parameter to PAL_TRUE when changing the port mode from active to passive or vice versa.

Return Values

RESULT_OK when function succeeds.

RESULT_ERROR when function fails.

-1 when the info member of mem_ifp is NULL or there is an internal error

NSM_LACP_ERROR_MAX_AGGREGATORS when the maximum number of aggregators have already been created

NSM_DRNI_ERROR_MLAG_MEM_INTF when LACP is part of an MLAG member

NSM_DRNI_ERROR_AGG_IPP when the port channel is part of an MLAG intra portal port

NSM_LACP_ERROR_FLOWCONTROL_MISMATCH when there is a flow control mismatch

NSM_LACP_ERROR_STORM_CONTROL_MISMATCH when there is a storm control mismatch

NSM_LACP_ERROR_ADMIN_KEY_MISMATCH when there is an administrative key mismatch

NSM_LACP_ERROR_BRIDGE_MISMATCH when there is a bridge mismatch

NSM_LACP_VLAN_PORT_MODE_MISMATCH when there is a port mode mismatch

NSM_LACP_ERROR_VLAN_MISMATCH when there is a VLAN mismatch

nsm_lacp_api_delete_aggregator_member

This function deletes a link aggregation group

This function is called by this command:

```
no channel-group
```

Syntax

```
#include "nsm/lacp/nsm_lacp.h"
int
nsm_lacp_api_delete_aggregator_member (struct nsm_master *nm,
                                       struct interface *mem_ifp,
                                       bool_t notify_lacp)
```

Input Parameters

nm	Pointer to NSM master.
mem_ifp	Pointer to the interface.
notify_lacp	Whether to notify LACP.

Return Values

RESULT_OK when function succeeds.

-1 when the info member of mem_ifp is NULL or there is an internal error

NSM_DRNI_ERROR_MLAG_MEM_INTF when LACP is part of an MLAG member

NSM_LACP_IF_CONSISTS_PROTECTION_GRP when the interface is part of a protection group

nsm_mlag_api_add_aggregator_member

This function assigns an MLAG identifier to an interface.

This function is called by this command:

```
mlag <1-65535>
```

Syntax

```
#include "nsm/mlag/drni/nsm_drni.h"
s_int32_t
nsm_mlag_api_add_aggregator_member (struct nsm_master *nm,
                                    struct interface *mem_ifp,
                                    u_int32_t key,
                                    u_char agg_type)
```

Input Parameters

`nm`	NSM master
`mem_ifp`	Port channel interface
`key`	MLAG identifier; this function adds an ”mlag” prefix to this value
`agg_type`	Aggregation type; this constant from `lib/nsm_message.h`: `AGG_CONFIG_MLAG`

Output Parameters

None

Return Values

0 when the function succeeds

NSM_DRNI_ERROR_MLAG_MEM_INTF_NOT_FOUND when the member interface cannot be found

NSM_DRNI_ERROR_CONFIGURED when `agg_type` is not valid or the member interface not mapped

NSM_DRNI_ERROR_MAX_AGGREGATORS when the maximum number of MLAG instances (`NSM_MAX_MLAG_AGGREGATORS`) has been created

NSM_DRNI_ERROR_MLAG_INTF_NOT_FOUND when the aggregation interface cannot be found

-1 when there is an internal error

NSM_DRNI_ERROR_MAX_AGGREGATOR_LINKS when the maximum number of aggregator links (`NSM_MAX_MLAG_AGGREGATOR_LINKS`) has been created

nsm_mlag_api_delete_aggregator_member

This function removes an MLAG identifier from an interface.

This function is called by this command:

```
no mlag
```

Syntax

```
#include "nsm/mlag/drni/nsm_drni.h"
s_int32_t
nsm_mlag_api_delete_aggregator_member (struct nsm_master *nm,
```

```
                                        struct interface *mem_ifp)
```

Input Parameters

`nm`	NSM master
`mem_ifp`	Interface

Output Parameters

None

Return Values

0 when the function succeeds

NSM_DRNI_ERROR_MLAG_MEM_INTF_NOT_FOUND when the member interface cannot be found

NSM_DRNI_ERROR_UNKNOWN when there is an internal error

NSM_DRNI_ERROR_MLAG_INTF_NOT_FOUND when the aggregation interface cannot be found

-1 when there is an internal error

LACP MIB Support

The LACP MIB is implemented according to the IEEE 802.1AX-REV-D3.1 MIB. The following subsections list the supported LACP MIB objects in the standard.

dot3adAggGroup

Objects in this group provide information about an aggregation.

Object Type	Syntax	Access
dot3adAggActorSystemID	MacAddress	read-only
dot3adAggActorSystemPriority	INTEGER	read-write
dot3adAggAggregateOrIndividual	TruthValue	read-only
dot3adAggActorAdminKey	INTEGER	read-write
dot3adAggMACAddress	StringOctet	read-only
dot3adAggActorOperKey	INTEGER	read-only
dot3adAggPartnerSystemID	MacAddress	read-only
dot3adAggPartnerSystemPriority	INTEGER	read-only
dot3adAggPartnerOperKey	INTEGER	read-only
dot3adAggCollectorMaxDelay	INTEGER	read-write

dot3adAggPortDebugGroup

Objects in this group provide debug information about every aggregated port.

Object Type	Syntax	Access
dot3adAggPortDebugRxState	INTEGER	read-only
dot3adAggPortDebugLastRxTime	TimeTicks	read-only
dot3adAggPortDebugMuxState	INTEGER	read-only
dot3adAggPortDebugMuxReason	String or StringOctet	read-only
dot3adAggPortDebugActorChurnState	INTEGER	read-only
dot3adAggPortDebugPartnerChurnState	INTEGER	read-only
dot3adAggPortDebugActorChurnCount	Counter32	read-only
dot3adAggPortDebugPartnerChurnCount	Counter32	read-only
dot3adAggPortDebugActorSyncTransitionCount	Counter32	read-only
dot3adAggPortDebugPartnerSyncTransitionCount	Counter32	read-only
dot3adAggPortDebugActorChangeCount	Counter32	read-only
dot3adAggPortDebugPartnerChangeCount	Counter32	read-only

dot3adAggPortGroup

Objects in this group provide information about every port in an aggregation.

Object Type	Syntax	Access
dot3adAggPortIndex	INTEGER	read-only
dot3adAggPortActorSystemPriority	INTEGER	read-write
dot3adAggPortActorSystemID	OctetStr	read-only
dot3adAggPortActorAdminKey	INTEGER	read-write
dot3adAggPortActorOperKey	INTEGER	read-only
dot3adAggPortPartnerAdminSystemPriority	INTEGER	read-write
dot3adAggPortPartnerOperSystemPriority	INTEGER	read-only
dot3adAggPortPartnerAdminSystemID	MacAddress	read-only
dot3adAggPortPartnerOperSystemID	MacAddress	read-only
dot3adAggPortPartnerAdminKey	INTEGER	read-write

dot3adAggPortPartnerOperKey	INTEGER	read-only
dot3adAggPortSelectedAggID	InterfaceIndex	read-only
dot3adAggPortAttachedAggID	InterfaceIndex	read-only
dot3adAggPortActorPort	INTEGER	read-only
dot3adAggPortActorPortPriority	INTEGER	read-write
dot3adAggPortPartnerAdminPort	INTEGER	read-write
dot3adAggPortPartnerOperPort	INTEGER	read-only
dot3adAggPortPartnerAdminPortPriority	INTEGER	read-write
dot3adAggPortPartnerOperPortPriority	INTEGER	read-only
dot3adAggPortActorAdminState	INTEGER	read-write Note: Writing to aggregation bit is not supported
dot3adAggPortActorOperState	INTEGER	read-only
dot3adAggPortPartnerAdminState	INTEGER	read-only
dot3adAggPortPartnerOperState	INTEGER	read-only
dot3adAggPortAggregateOrIndividual	TruthValue	read-only

dot3adAggPortListGroup

Objects in this group provide information about every port in an aggregation.

Object Type	Syntax	Access
dot3adAggPortListPorts	PortList	read-only

dot3adAggPortStatsGroup

Objects in this group provide information about every port in an aggregation.

Object Type	Syntax	Access
dot3adAggPortStatsLACPDUsRx	Counter32	read-only
dot3adAggPortStatsMarkerPDUsRx	Counter32	read-only
dot3adAggPortStatsIllegalRx	Counter32	read-only
dot3adAggPortStatsLACPDUsTx	Counter32	read-only
dot3adAggPortStatsMarkerPDUsTx	Counter32	read-only

MIB Functions

This section describes functions related to reading and writing MIBs.

lacp_snmp_dot3adAggPortDebugTable

This function is a callback procedure for the lacp_snmp_dot3adAggPortTable function (see lacp_snmp_dot3adAggPortTable on page 221).

Syntax

```
unsigned char *
lacp_snmp_dot3adAggPortDebugTable(struct variable *vp,
                                  oid * name,
                                  size_t * length,
                                  int exact,
                                  size_t * var_len,
                                  WriteMethod ** write_method,
                                  u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

lacp_snmp_dot3adAggPortListTable

This function is made when an external network management system queries an SNMP variable in dot3adAggPortListGroup (see dot3adAggPortListGroup on page 218).

Syntax

```
unsigned char *
lacp_snmp_dot3adAggPortListTable(struct variable *vp, oid * name,
                                 size_t * length, int exact,
                                 size_t * var_len,
                                 WriteMethod ** write_method,
                                 u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure.
name	Pointer to the variable name (OID).
length	Number of elements (sub-ids) in the name.
exact	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
vr_id	Unused.

Output Parameters

var_len	Length of the variable that was read.
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

lacp_snmp_dot3adAggPortStatsTable

This function is made when an external network management system queries an SNMP variable in dot3adAggPortStatsGroup (see dot3adAggPortStatsGroup on page 218).

Syntax

```
unsigned char *
lacp_snmp_dot3adAggPortStatsTable(struct variable *vp, oid * name,
    size_t * length, int exact,
    size_t * var_len, WriteMethod ** write_method,
    u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure.
name	Pointer to the variable name (OID).
length	Number of elements (sub-ids) in the name.
exact	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
vr_id	Unused.

Output Parameters

var_len	Length of the variable that was read.
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

lacp_snmp_dot3adAggPortTable

This function is made when an external network management system queries an SNMP variable in dot3adAggPortGroup (see dot3adAggPortGroup on page 217).

Syntax

```
unsigned char *
lacp_snmp_dot3adTablesLastChanged (struct variable *vp,
                                   oid * name,
                                   size_t * length,
                                   int exact,
                                   size_t * var_len,
                                   WriteMethod ** write_method,
                                   u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).
`length`	Number of elements (sub-ids) in the name.
`exact`	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
`vr_id`	Unused.

Output Parameters

`var_len`	Length of the variable that was read.
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable when this function succeeds

0 or NULL when this function fails

lacp_snmp_dot3adAggTable

This function is made when an external network management system queries an SNMP variable in dot3adAggGroup (see dot3adAggPortGroup on page 217).

Syntax

```
unsigned char *
lacp_snmp_dot3adAggTable (struct variable *vp,
                          oid * name,
                          size_t * length,
                          int exact,
                          size_t * var_len, WriteMethod ** write_method,
                          u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure.
`name`	Pointer to the variable name (OID).

length	Number of elements (sub-ids) in the name.
exact	Whether this request is a GET (exact match: 1 = PAL_TRUE) or a GETNEXT (0 = PAL_FALSE).
vr_id	Unused.

Output Parameters

var_len	Length of the variable that was read.
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable.

Return Value

A pointer to the value of the variable if successful.

0 or NULL if not successful.

lacp_snmp_write_dot3adAggPortTable

This function is called when an external network management system sets an SNMP variable in dot3adAggPortGroup (see dot3adAggPortGroup on page 217).

Syntax

```
int
lacp_snmp_write_dot3adAggPortTable_States (int action, u_char * var_val,
                                           u_char var_val_type,
                                           size_t var_val_len,
                                           u_char * statP, oid * name,
                                           size_t length,
                                           struct variable *vp,
                                           u_int32_t vr_id)
```

Input Parameters

action	Unused.
var_val	Value of the variable.
var_val_type	Data type of the variable; this constant from lib/asn1.h. It corresponds to ASN.1's built-in simple type: ASN_INTEGER
var_val_len	Length of the variable.
statP	Unused.
name	Pointer to variable name (OID).
length	Length of the name.
vp	Pointer to SNMP variable structure.
vr_id	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

lacp_snmp_write_dot3adAggPortTable_States

This function is called when an external network management system sets an SNMP variable in dot3adAggPortStatsGroup (see dot3adAggPortStatsGroup on page 218).

Syntax

```
int
lacp_snmp_write_dot3adAggPortTable_States (int action, u_char * var_val,
                                           u_char var_val_type,
                                           size_t var_val_len,
                                           u_char * statP, oid * name,
                                           size_t length,
                                           struct variable *vp,
                                           u_int32_t vr_id)
```

Input Parameters

`action`	Unused.
`var_val`	Value of the variable.
`var_val_type`	Data type of the variable; this constant from lib/asn1.h. It corresponds to ASN.1's built-in simple type: ASN_OCTET_STR
`var_val_len`	Length of the variable.
`statP`	Unused.
`name`	Pointer to variable name (OID).
`length`	Length of the name.
`vp`	Pointer to SNMP variable structure.
`vr_id`	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

lacp_snmp_write_dot3adAggTable

This function is called when an external network management system sets an SNMP variable in dot3adAggGroup (see dot3adAggGroup on page 216).

Syntax

```
int
lacp_snmp_write_dot3adAggTable (int action, u_char * var_val,
                                u_char var_val_type, size_t var_val_len,
                                u_char * statP, oid * name, size_t length,
                                struct variable *vp, u_int32_t vr_id)
```

Input Parameters

`action`	Unused.
`var_val`	Value of the variable.
`var_val_type`	Data type of the variable; this constant from lib/asn1.h. It corresponds to ASN.1's built-in simple type: ASN_INTEGER.
`var_val_len`	Length of the variable.
`statP`	Unused.
`name`	Pointer to variable name (OID).
`length`	Length of the name.
`vp`	Pointer to SNMP variable structure.
`vr_id`	Unused.

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when function succeeds.

SNMP_ERR_GENERR when function fails.

SNMP_ERR_BADVALUE when the `var_val` is invalid

SNMP_ERR_WRONGTYPE when the `var_val_type` is invalid

SNMP_ERR_WRONGLENGTH when the `var_val_len` is invalid

CHAPTER 11 Remote Monitoring

This chapter describes the API and SNMP support for Remote Monitoring.

Data Structures

This data structure is used by multiple ZebOS-XP modules and is documented in the *Common Data Structures Developer Guide*:

- `prefix`

rmon_master

This data structure in `rmond/rmon_config.h` is the RMON master.

Member	Description
`vr`	Pointer to VR
`zg`	Pointer to lib_globals
`etherStats_table`	Collection of stat entries in route table
`historyControl_table`	History control table
`alarm_table`	Alarm entry table
`event_table`	Event entry table

Definition

```
struct rmon_master
{
  /* Pointer to VR. */
  struct ipi_vr *vr;
  /* pointer to lib_globals */
  struct lib_globals *zg;
#define RMON_ZG   (rmonzg)
  struct route_table *etherStats_table;
  struct route_table *historyControl_table;
/*  struct route_table *historyStats_table;*/
  struct route_table *alarm_table;
  struct route_table *event_table;
}
```

rmon_AlarmGroup

This data structure in `rmond/rmon_config.h` is an RMON alarm group.

Member	Description
`alarmIndex`	An index that uniquely identifies an entry in the alarm table
`alarmInterval`	Interval in seconds over which the data is sampled and compared with the rising and falling thresholds
`alarmVariable`	Object identifier of the variable to be sampled
`alarmConf`	
`alarmVariableWord`	Alarm variable word TLV
`alarmSampleType`	1 - absoluteValue, 2 - deltaValue
`alarmValue`	Value of the statistic during the last sampling period
`alarmStartupAlarm`	Alarm that may be sent when this entry is first set to valid
`alarmRisingThreshold`	When the current sampled value is greater than or equal to this threshold, and the value at the last sampling interval was less than this threshold, a single event is generated
`alarmFallingThreshold`	When the current sampled value is less than or equal to this threshold, and the value at the last sampling interval was greater than this threshold, a single event is generated
`alarmRisingEventIndex`	Index of the eventEntry that is used when a rising threshold is crossed
`alarmFallingEventIndex`	Index of the eventEntry that is used when a falling threshold is crossed
`alarmOwner`	Entity that configured this entry and is using the resources assigned to it
`alarmStatus`	1 - Valid 2 - CreateRequest 3 - UnderCreation 4 - Invalid 5 - Non-existing
`alarm_timer`	Alarm timer for RMON
`alarm_prev_value`	Previous value of the alarm
`alarm_curr_value`	Current value of the alarm
`alarm_last_event`	Last value of the alarm
`alarm_startup_alarm_status`	Startup alarm status of RMON

Definition

```
struct rmon_AlarmGroup
{
```

```
  u_int32_t alarmIndex;
#define RMON_ALARM_INTERVAL              (1 << 0)
#define RMON_ALARM_VARIABLE              (1 << 1)
#define RMON_ALARM_SAMPLETYPE            (1 << 2)
#define RMON_ALARM_STARTUP_ALARM         (1 << 3)
#define RMON_ALARM_RISING_THRESHOLD      (1 << 4)
#define RMON_ALARM_FALLING_THRESHOLD     (1 << 5)
#define RMON_ALARM_RISING_EVENT_IDX      (1 << 6)
#define RMON_ALARM_FALLING_EVENT_IDX     (1 << 7)
#define RMON_ALARM_OWNER                 (1 << 8)
#define RMON_ALARM_STATUS                (1 << 9)
#define RMON_ALARM_VARIABLE_MAXSIZE      13
#define ETHERSTATSENTRY                  "etherStatsEntry."
#define ETHERSTATSNUM                    "1.3.6.1.2.1.16.1.1.1"
  s_int32_t alarmInterval;
#ifdef HAVE_SNMP
  oid       alarmVariable[RMON_ALARM_VARIABLE_MAXSIZE];
#endif /* HAVE_SNMP */
#define RMON_ALARM_OWNER_CONF            (1 << 0)
  u_int8_t alarmConf;
  char alarmVariableWord[RMON_ALARM_VAR_WORD_LENGTH+1];
  u_int32_t alarmSampleType; /*1- absoluteValue, 2- deltaValue */
  ut_int64_t alarmValue;
  u_int32_t alarmStartupAlarm;
  ut_int64_t alarmRisingThreshold; /* For ether stats type is ut_int64_t */
  ut_int64_t alarmFallingThreshold;
  u_int32_t alarmRisingEventIndex;
  u_int32_t alarmFallingEventIndex;
  char      alarmOwner[RMON_OWNER_NAME_SIZE + 1];
  u_int32_t alarmStatus;      /*1-Valid, 2-createRequest,
                               *3-UnderCreation, 4- invalid,
                               *5-nonExistent */
  struct thread *alarm_timer;
  ut_int64_t alarm_prev_value;
  ut_int64_t alarm_curr_value;
  u_int16_t alarm_last_event;
  u_int16_t alarm_startup_alarm_status;
};
```

rmon_etherStatsGroup

The members in this data structure in `rmond/rmon_config.h` correspond to objects with the same name in the etherStatsTable defined by RFC 2819.

Definition

```
struct rmon_etherStatsGroup
{
  u_int32_t etherStatsIndex;
#define RMON_ETHER_STATS_DATASOURCE      (1 << 0)
#define RMON_ETHER_STATS_OWNER           (1 << 1)
```

```
#define RMON_ETHER_STATS_STATUS          (1 << 2)

  oid       etherStatsDataSource[RMON_DATA_SOURCE_MAXSIZE];
  ut_int64_t etherStatsDropEvents;
  ut_int64_t etherStatsOctets;
  ut_int64_t etherStatsPkts;
  ut_int64_t etherStatsBroadcastPkts;
  ut_int64_t etherStatsMulticastPkts;
  ut_int64_t etherStatsCRCAlignErrors;
  ut_int64_t etherStatsUndersizePkts;
  ut_int64_t etherStatsOversizePkts;
  ut_int64_t etherStatsFragments;
  ut_int64_t etherStatsJabbers;
  ut_int64_t etherStatsCollisions;
  ut_int64_t etherStatsPkts64Octets;
  ut_int64_t etherStatsPkts65to127Octets;
  ut_int64_t etherStatsPkts128to255Octets;
  ut_int64_t etherStatsPkts256to511Octets;
  ut_int64_t etherStatsPkts521to1023Octets;
  ut_int64_t etherStatsPkts1024to1518Octets;
  char      *etherStatsOwner;
  u_int32_t etherStatsStatus; /*1-Valid, 2-createRequest,
                               *3-UnderCreation, 4- invalid,
                               *5-nonExistent */
};
```

rmon_EventGroup

This data structure in `rmond/rmon_config.h` is an RMON event group.

Member	Description
`eventIndex`	Index that uniquely identifies an entry in the event table
`eventConf`	
`eventDescription`	Comment describing this event entry
`eventType`	1 - None 2 - Log 3 - SNMP rap 4 - Log and trap
`eventCommunity`	If an SNMP trap is to be sent, it will be sent to this SNMP community
`eventLastTimeSent`	Value of sysUpTime at the time this event entry last generated an event

Member	Description
eventOwner	Entity that configured this entry and is using the resources assigned to it
eventStatus	1 - Valid 2 - CreateRequest 3 - UnderCreation 4 - Invalid 5 - Non-existing

Definition

```
struct rmon_EventGroup
{
  u_int32_t eventIndex;

#define RMON_EVENT_CONF_DESCRIPTION     (1 << 0)
#define RMON_EVENT_CONF_OWNER           (1 << 1)
  u_int8_t eventConf;

  char     eventDescription[RMON_DESCR_LENGTH + 1];
  u_int32_t eventType;        /*1- none, 2- log, 3-snmpTrap, 4-logandTrap */

  char     eventCommunity[RMON_COMM_LENGTH + 1];
  s_int32_t eventLastTimeSent;

  char     eventOwner [RMON_OWNER_NAME_SIZE + 1];
  u_int32_t eventStatus;      /*1-Valid, 2-createRequest,
                               *3-UnderCreation, 4- invalid,
                               *5-nonExistent */
};
```

rmon_HistoryControlGroup

This data structure in `rmond/rmon_config.h` is an RMON history control group.

Member	Description
historyControlIndex	Index that uniquely identifies an entry in the historyControl table
historyControlDataSource	Source of the data (ifindex)
historyControlBucketsRequested	Requested number of discrete time intervals over which data is to be saved
historyControlBucketsGranted	Number of discrete sampling intervals over which data shall be saved
historyControlInterval	Interval in seconds over which the data is sampled for each bucket
historyControlOwner	Entity that configured this entry and is using the resources assigned to it

Member	Description
historyControlStatus	1 - Valid 2 - CreateRequest 3 - UnderCreation 4 - Invalid 5 - Non-existing
current_sample_no	Count of samples
rmon_coll_history_timer	Timer thread
historyStats_table	Historystat for the route table

Definition

```
struct rmon_HistoryControlGroup
{
  u_int32_t historyControlIndex;
#define RMON_HISTORY_CONTROL_DATASOURCE  (1 << 0)
#define RMON_HISTORY_CONTROL_BUCKETS_REQ (1 << 1)
#define RMON_HISTORY_CONTROL_INTERVAL    (1 << 2)
#define RMON_HISTORY_CONTROL_OWNER       (1 << 3)
#define RMON_HISTORY_CONTROL_STATUS      (1 << 4)

  u_int32_t historyControlDataSource; /* As of now it is ifindex */
  u_int32_t historyControlBucketsRequested;
  u_int32_t historyControlBucketsGranted;
  u_int32_t historyControlInterval;
  char      *historyControlOwner;
  u_int32_t historyControlStatus; /*1-Valid, 2-createRequest,
                                   *3-UnderCreation, 4- invalid,
                                   *5-nonExistent */
  /* count of samples */
  u_int32_t current_sample_no;

  /* Timer thread */
  struct thread *rmon_coll_history_timer;
  struct route_table *historyStats_table;
};
```

rmon_if_stats

This data structure is in `rmond/rmon_config.h`.

Member	Description
ifindex	Interface index
good_octets_rcv	Good octets received

Member	Description
bad_octets_rcv	Bad octets received
mac_transmit_err	MAC transmit errors
good_pkts_rcv	Good packets received
bad_pkts_rcv	Bad packets received
brdc_pkts_rcv	Broadcast packets received
mc_pkts_rcv	Multicast packets received
pkts_64_octets_rcv	Packets 64 octets in length received
pkts_65_127_octets_rcv	Packets 65-127 octets in length received
pkts_128_255_octets_rcv	Packets 128-255 octets in length received
pkts_256_511_octets_rcv	Packets 256 511 octets in length received
pkts_512_1023_octets_rcv	Packets 512-1023 octets in length received
pkts_1024_1518_octets_rcv	Packets 1024-1518 octets in length received
good_octets_sent	Good octets sent
good_pkts_sent	Good packets sent
excessive_collisions	Estimate of the total number of collisions on this Ethernet segment
mc_pkts_sent	Multicast packets sent
brdc_pkts_sent	Broadcast packets sent
unrecog_mac_cntr_rcv	Unrecognized MAC counter received
fc_sent	Flow control (PAUSE) frames sent
good_fc_rcv	Good flow control (PAUSE) frames received
drop_events	Number of times packets dropped due to a lack of resources
undersize_pkts	Number of packets received that were less than 64 octets
fragments_pkts	Fragments packets
oversize_pkts	Number of packets received that were longer than 1518 octets
jabber_pkts	Jabber packets
mac_rcv_error	MAC receive errors
bad_crc	Bad CRC
collisions	Collisions

Member	Description
late_collisions	Late collisions
bad_fc_rcv	Bad flow control (PAUSE) frames received

Definition

```
struct rmon_if_stats
{
  u_int32_t ifindex;
  ut_int64_t good_octets_rcv;
  ut_int64_t bad_octets_rcv;
  ut_int64_t mac_transmit_err;
  ut_int64_t good_pkts_rcv;
  ut_int64_t bad_pkts_rcv;
  ut_int64_t brdc_pkts_rcv;
  ut_int64_t mc_pkts_rcv;
  ut_int64_t pkts_64_octets_rcv;
  ut_int64_t pkts_65_127_octets_rcv;
  ut_int64_t pkts_128_255_octets_rcv;
  ut_int64_t pkts_256_511_octets_rcv;
  ut_int64_t pkts_512_1023_octets_rcv;
  ut_int64_t pkts_1024_1518_octets_rcv;
  ut_int64_t good_octets_sent;
  ut_int64_t good_pkts_sent;
  ut_int64_t excessive_collisions;
  ut_int64_t mc_pkts_sent;
  ut_int64_t brdc_pkts_sent;
  ut_int64_t unrecog_mac_cntr_rcv;
  ut_int64_t fc_sent;
  ut_int64_t good_fc_rcv;
  ut_int64_t drop_events;
  ut_int64_t undersize_pkts;
  ut_int64_t fragments_pkts;
  ut_int64_t oversize_pkts;
  ut_int64_t jabber_pkts;
  ut_int64_t mac_rcv_error;
  ut_int64_t bad_crc;
  ut_int64_t collisions;
  ut_int64_t late_collisions;
  ut_int64_t bad_fc_rcv;
};
```

API

This section contains the Remote Monitoring functions for ZebOS-XP.

Function	Description
rmon_alarm_entry_set	Adds an alarm entry
rmon_alarm_index_remove	Removes an alarm entry
rmon_collection_stat_entry_add	Adds a collection statistics entry on an interface
rmon_collection_stat_entry_remove	Removes a collection statistics entry on an interface
rmon_coll_history_bucket_set	Sets the buckets requested for a history control entry on an interface
rmon_coll_history_datasource_set	Associates a data source to the history control table
rmon_coll_history_index_remove	Removes an entry from the history control table
rmon_coll_history_index_set	Adds a collection history control entry on an interface
rmon_coll_history_interval_set	Sets the interval of the history control entry on an interface
rmon_coll_history_owner_set	Sets the owner of the history control entry on an interface
rmon_coll_history_set_active	Sets the history control entry to active status
rmon_coll_history_set_inactive	Sets the history control entry to inactive status
rmon_coll_history_validate	Checks if the history control parameters are already set on this interface
rmon_coll_stats_validate	Checks if the collection is already enabled on the interface
rmon_event_comm_set	Sets the community name of the event entry
rmon_event_description_set	Sets the description of the event entry
rmon_event_index_remove	Removes an event entry
rmon_event_index_set	Adds an event table entry
rmon_event_owner_set	Sets the owner name of the event entry
rmon_event_set_active	Makes an event table entry active
rmon_event_type_set	Sets the type of the event entry
rmon_nsm_send_req_statistics	Gets interface statistics
rmon_set_alarm_falling_event_index	Sets the event corresponding to crossing the falling threshold of the alarm
rmon_set_alarm_falling_threshold	Sets the falling threshold value of the alarm entry
rmon_set_alarm_interval	Sets the alarm polling interval
rmon_set_alarm_owner	Sets the owner of the alarm
rmon_set_alarm_rising_event_index	Sets the event corresponding to crossing the rising threshold of the alarm
rmon_set_alarm_rising_threshold	Sets the rising threshold value of the alarm entry
rmon_set_alarm_sample_type	Sets the sample type of the alarm entry

Function	Description
rmon_set_alarm_start_up	Sets the alarm start-up type of the alarm entry
rmon_set_alarm_variable	Sets the variable of the alarm entry
rmon_snmp_set_alarm_status	Sets the status of an entry in the alarmTable
rmon_snmp_set_ether_stats_status	Sets the status of an entry in the etherStatsTable
rmon_snmp_set_event_community	Sets the community name of an entry in the eventTable
rmon_snmp_set_event_description	Sets the description of an entry in the eventTable
rmon_snmp_set_event_owner	Sets the owner name of an entry in the eventTable
rmon_snmp_set_event_status	Sets the status of an entry in the eventTable
rmon_snmp_set_event_type	Sets the type of an entry in the eventTable
rmon_snmp_set_history_status	Sets the status of an entry in the historyControlTable

Include File

To call the functions in this chapter, you must include `rmond/rmon_api.h`.

rmon_alarm_entry_set

This function adds an alarm entry.

Syntax

```
s_int32_t
rmon_alarm_entry_set (u_int32_t alarm_index, char *Variable,
u_int32_t interval, u_int32_t RisingValue, u_int32_t FallingValue,
u_int32_t rising_event, u_int32_t falling_event, char *ownername);
```

Input Parameters

`alarm_index`	Alarm entry index
`variable`	Variable OID name
`interval`	EtherStats entry index
`risingValue`	Rising threshold value
`fallingValue`	Falling threshold value
`rising_event`	Rising event value
`falling_event`	Falling event value
`ownername`	Owner name

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR or RMON_API_SET_FAILURE when the function fails

rmon_alarm_index_remove

This function removes an alarm entry.

Syntax

```
s_int32_t
rmon_alarm_index_remove (u_int32_t alarm_index);
```

Input Parameters

`alarm_index`	Alarm entry index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be removed

rmon_collection_stat_entry_add

This function adds a collection statistics entry on an interface.

Syntax

```
s_int32_t
rmon_collection_stat_entry_add (u_int32_t ifindex, u_int32_t index,
char * ownername);
```

Input Parameters

`index`	Etherstats entry index
`ifindex`	Interface index
`ownername`	Owner name

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be added

rmon_collection_stat_entry_remove

This function removes a collection statistics entry on an interface.

Syntax

```
s_int32_t
rmon_collection_stat_entry_remove (u_int32_t ifindex, u_int32_t index);
```

Input Parameters

`index`	Etherstats entry index
`ifindex`	Interface index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be removed

rmon_coll_history_bucket_set

This function sets the buckets requested for a history control entry on an interface.

Syntax

```
s_int32_t
rmon_coll_history_bucket_set (u_int32_t index, u_int32_t bucketno);
```

Input Parameters

`index`	History control entry index
`bucketno`	Number of buckets

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

rmon_coll_history_datasource_set

This function associates a data source to the history control table.

Syntax

```
s_int32_t
rmon_coll_history_datasource_set (u_int32_t index, oid *name)
```

Input Parameters

`index`	History control entry index
`name`	Object identifier of the data source

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR or RMON_API_SET_FAILURE when the function fails

rmon_coll_history_index_remove

This function removes an entry from the history control table.

Syntax

```
s_int32_t
rmon_coll_history_index_remove (u_int32_t index);
```

Input Parameters

`index`	History control entry index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be removed

rmon_coll_history_index_set

This function adds a collection history control entry on an interface.

Syntax

```
s_int32_t
rmon_coll_history_index_set (u_int32_t ifindex, u_int32_t index);
```

Input Parameters

`index`	History control entry index
`ifindex`	Interface index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be added

rmon_coll_history_interval_set

This function sets the interval of the history control entry on an interface.

Syntax

```
s_int32_t
rmon_coll_history_interval_set (u_int32_t index, u_int32_t interval);
```

Input Parameters

`index`	History control entry index
`interval`	Polling interval

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

rmon_coll_history_owner_set

This function sets the owner of the history control entry on an interface.

Syntax

```
s_int32_t
rmon_coll_history_owner_set(u_int32_t index, char * ownername);
```

Input Parameters

`index`	History control entry index
`ownername`	Owner

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the function fails

rmon_coll_history_set_active

This function sets the history control entry to active status.

Syntax

```
s_int32_t
rmon_coll_history_set_active (u_int32_t index);
```

Input Parameters

`index`	History control entry index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be set to the active state

rmon_coll_history_set_inactive

This function sets the history control entry to inactive status.

Syntax

```
s_int32_t
rmon_coll_history_set_inactive (u_int32_t index);
```

Input Parameters

`index`	History control entry index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be set to the inactive state

rmon_coll_history_validate

This function checks if the history control parameters are already set on this interface.

Syntax

```
s_int32_t
rmon_coll_history_validate (u_int32_t index, u_int32_t ifindex);
```

Input Parameters

`index`	History control entry index
`ifindex`	Interface index

Output Parameters

None

Return Value

Zero (0) when the function succeeds

RESULT_ERROR when the entry is already present

rmon_coll_stats_validate

This function checks if the collection is already enabled on the interface.

Syntax

```
s_int32_t
rmon_coll_stats_validate (u_int32_t index, u_int32_t ifindex);
```

Input Parameters

`index`	Etherstats entry index
`ifindex`	Interface index

Output Parameters

None

Return Value

Zero (0) when the function succeeds

RESULT_ERROR when the collection entry is already present

rmon_event_comm_set

This function sets the community name of the event entry.

Syntax

```
s_int32_t
rmon_event_comm_set (u_int32_t index, char *comm);
```

Input Parameters

`index`	Event entry index
`comm`	Trap community

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_event_description_set

This function sets the description of the event entry.

Syntax

```
s_int32_t
rmon_event_description_set (u_int32_t index, char *desc);
```

Input Parameters

`index`	Event entry index
`desc`	Event description

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_event_index_remove

This function removes an event entry.

Syntax

```
s_int32_t
rmon_event_index_remove (u_int32_t index);
```

Input Parameters

`index`	Event entry index

Output Parameters

None

Return Value

RESULT_OK when the function succeeds

RESULT_ERROR when the entry cannot be removed

rmon_event_index_set

This function adds an event table entry.

Syntax

```
s_int32_t
rmon_event_index_set (u_int32_t index);
```

Input Parameters

`index`	Event entry index

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the `rmon_master` is NULL

rmon_event_owner_set

This function sets the owner name of the event entry.

Syntax

```
s_int32_t
rmon_event_owner_set (u_int32_t index, char *owner);
```

Input Parameters

`index`	Event entry index
`owner`	Owner name

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_event_set_active

This function makes an event table entry active.

Syntax

```
s_int32_t
rmon_event_set_active (u_int32_t index);
```

Input Parameters

`index`	Event entry index

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_event_type_set

This function sets the type of the event entry.

Syntax

```
s_int32_t
rmon_event_type_set (u_int32_t index, u_int32_t type);
```

Input Parameters

`index`	Event entry index

type	Type:
1	None
2	Log
3	SNMP trap
4	Log and trap

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_nsm_send_req_statistics

This function gets interface statistics.

Syntax

```
s_int32_t
rmon_nsm_send_req_statistics (u_int32_t ifindex, struct rmon_if_stats *if_stats)
```

Input Parameters

ifindex	Interface index

Output Parameters

if_stats	Pointer to interface statistics struct

Return Value

RESULT_OK or zero (0) when the function succeeds

rmon_set_alarm_falling_event_index

This function sets the event corresponding to crossing the falling threshold value of the alarm entry.

Syntax

```
s_int32_t
rmon_set_alarm_falling_event_index (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t event_ix);
```

Input Parameters

rm	Pointer to the RMON master
rmon_alarm	Pointer to the alarm entry
index	Alarm entry index
event_ix	Falling event value

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_falling_threshold

This function sets the falling threshold value of the alarm entry.

Syntax

```
s_int32_t
rmon_set_alarm_falling_threshold (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t falling_th);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`falling_th`	Falling threshold value

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_interval

This function sets the alarm polling interval.

Syntax

```
s_int32_t
rmon_set_alarm_interval (osti rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t interval);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`interval`	Polling interval

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_owner

This function sets the owner of the alarm.

Syntax

```
s_int32_t
rmon_set_alarm_owner (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, char *owner);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`ownername`	Owner

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_rising_event_index

This function sets the event corresponding to crossing the rising threshold value of the alarm entry.

Syntax

```
s_int32_t rmon_set_alarm_rising_event_index (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t event_ix);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`event_ix`	Rising event value

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_rising_threshold

This function sets the rising threshold value of the alarm entry.

Syntax

```
s_int32_t rmon_set_alarm_rising_threshold (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t rising_th);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`rising_th`	Rising threshold value

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_sample_type

This function sets the sample type of the alarm entry.

Syntax

```
s_int32_t
rmon_set_alarm_sample_type (struct rmon_master *rm, struct rmon_AlarmGroup *rmon_alarm,
u_int32_t index, u_int32_t sample_type);
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`sample_type`	Alarm sample type (Absolute -1, Delta -2)

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_start_up

This function sets the alarm start-up type of the alarm entry.

Syntax

```
s_int32_t
rmon_set_alarm_start_up (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, u_int32_t startup)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`startup`	Start-up alarm type (rising alarm -1, falling alarm - 2, rising or falling alarm - 3)

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_set_alarm_variable

This function sets the variable of the alarm entry.

Syntax

```
s_int32_t
rmon_set_alarm_variable (struct rmon_master *rm,
struct rmon_AlarmGroup *rmon_alarm, u_int32_t index, oid *name)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm entry
`index`	Alarm entry index
`name`	Variable object identifier (OID)

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_alarm_status

This function sets the status of an entry in the alarmTable.

Syntax

```
s_int32_t
rmon_snmp_set_alarm_status (struct rmon_master *rm, struct rmon_AlarmGroup *rmon_alarm,
u_int32_t index, u_int32_t status)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_alarm`	Pointer to the alarm group
`index`	Alarm entry index
`status`	The status to set; one of the constants below from `rmond/rmon_config.h`. See the EntryStatus syntax item in RFC 2819 for valid state transitions:

`VALID_STATUS`

`CREATE_REQ_STATUS`

`UNDER_CREATION_STATUS`

`INVALID_STATUS`

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_ether_stats_status

This function sets the status of an entry in the etherStatsTable.

Syntax

```
s_int32_t
rmon_snmp_set_ether_stats_status (struct rmon_master *rm,
struct rmon_etherStatsGroup *rmon_ether_stats, u_int32_t index, u_int32_t status)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_ether_stats`	Pointer to the etherStatsTable
`index`	Status entry index
`status`	The status to set; one of the constants below from rmond/rmon_config.h. See the EntryStatus syntax item in RFC 2819 for valid state transitions:

`VALID_STATUS`

`CREATE_REQ_STATUS`

UNDER_CREATION_STATUS

INVALID_STATUS

Output Parameters

None.

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_event_community

This function sets the community name of an entry in the eventTable.

Syntax

```
s_int32_t
rmon_snmp_set_event_community (struct rmon_master *rm,
struct rmon_EventGroup *rmon_event, u_int32_t index, char *comm)
```

Input Parameters

rm	Pointer to the RMON master
rmon_event	Pointer to the event entry
index	Event entry index
comm	Community name

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_event_description

This function sets the description of an entry in the eventTable.

Syntax

```
s_int32_t
rmon_snmp_set_event_description(struct rmon_master *rm,
struct rmon_EventGroup *rmon_event, u_int32_t index, char *descr)
```

Input Parameters

rm	Pointer to the RMON master
rmon_event	Pointer to the event entry
index	Event entry index

`descr`	Event description

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_event_owner

This function sets the owner name of an entry in the eventTable.

Syntax

```
s_int32_t
rmon_snmp_set_event_owner (struct rmon_master *rm,
struct rmon_EventGroup *rmon_event, u_int32_t index, char *owner)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_event`	Pointer to the event entry
`index`	Event entry index
`owner`	Owner name

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_event_status

This function sets the status of an entry in the eventTable.

Syntax

```
s_int32_t
rmon_snmp_set_event_status (struct rmon_master *rm, struct rmon_EventGroup *rmon_event,
u_int32_t index, u_int32_t status)
```

Input Parameters

`rm`	Pointer to the RMON master
`rmon_event`	Pointer to the event
`index`	Event entry index
`status`	The status to set; one of the constants below from `rmond/rmon_config.h`. See the EntryStatus syntax item in RFC 2819 for valid state transitions:

VALID_STATUS

CREATE_REQ_STATUS

UNDER_CREATION_STATUS

INVALID_STATUS

Output Parameters

None.

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_event_type

This function sets the type of an entry in the eventTable.

Syntax

```
s_int32_t
rmon_snmp_set_event_type (struct rmon_master *rm,
struct rmon_EventGroup *rmon_event, u_int32_t index, u_int32_t type)
```

Input Parameters

rm	Pointer to the RMON master
rmon_event	Pointer to the event entry
index	Event entry index
type	Type of event:
1	None
2	Log
3	SNMP trap
4	Log and trap

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS when the function succeeds

RMON_API_SET_FAILURE when the function fails

rmon_snmp_set_history_status

This function sets the status of an entry in the historyControlTable.

Syntax

```
s_int32_t
```

```
rmon_snmp_set_history_status (struct rmon_master *rm,
struct rmon_HistoryControlGroup *rmon_history, u_int32_t index, u_int32_t status)
```

Input Parameters

`rm`	Pointer to RMON master
`rmon_history`	Pointer to RMON history control group
`index`	Index of the history control entry
`status`	The status to set; one of the constants below from `rmond/rmon_config.h`. See the EntryStatus syntax item in RFC 2819 for valid state transitions:

```
VALID_STATUS
CREATE_REQ_STATUS
UNDER_CREATION_STATUS
INVALID_STATUS
```

Output Parameters

None

Return Value

RMON_API_SET_SUCCESS or RESULT_OK when the function succeeds

RMON_API_SET_FAILURE or RESULT_ERROR when the function fails

Remote Monitoring MIB

The RMON MIB is implemented according to RFC 2819 (Remote Network Monitoring MIB).

Supported Tables

The sections below list the MIB objects in the RFC and, where relevant, show supporting functions in ZebOS-XP.

etherStatsTable

Objects in this table store statistics for an Ethernet interface.

Supported objects:

Object Type	Syntax	Access	Functions
etherStatsIndex	Integere32	read-only	None
etherStatsDataSource	OBJECT IDENTIFIER	read-create	None
etherStatsDropEvents	Counter32	read-only	None
etherStatsOctets	Counter32	read-only	None
etherStatsPkts	Counter32	read-only	None
etherStatsBroadcastPkts	Counter32	read-only	None

Object Type	Syntax	Access	Functions
etherStatsMulticastPkts	Counter32	read-only	None
etherStatsCRCAlignErrors	Counter32	read-only	None
etherStatsUndersizePkts	Counter32	read-only	None
etherStatsOversizePkts	Counter32	read-only	None
etherStatsFragments	Counter32	read-only	None
etherStatsJabbers	Counter32	read-only	None
etherStatsPkts64Octets	Counter32	read-only	None
etherStatsPkts65to127Octets	Counter32	read-only	None
etherStatsPkts128to255Octets	Counter32	read-only	None
etherStatsPkts256to511Octets	Counter32	read-only	None
etherStatsPkts512to1023Octets	Counter32	read-only	None
etherStatsPkts1024to1518Octets	Counter32	read-only	None
etherStatsOwner	OwnerString	read-create	None
etherStatsStatus	EntryStatus	read-create	rmon_snmp_set_ether_stats_status

Unsupported objects:

- etherStatsCollisions

historyControlTable

Objects in this table store parameters that set up a periodic sampling of statistics.

Supported objects:

Object Type	Syntax	Access	Functions
historyControlIndex	Integer32	read-only	None
historyControlDataSource	OBJECT IDENTIFIER	read-create	rmon_coll_history_datasource_set
historyControlBucketsRequested	Interger32	read-create	rmon_coll_history_bucket_set
historyControlBucketsGranted	Integer32	read-only	None
historyControlInterval	Integer32	read-create	rmon_coll_history_interval_set
historyControlOwner	OwnerString	read-create	rmon_coll_history_owner_set
historyControlStatus	EntryStatus	read-create	rmon_snmp_set_history_status

etherHistoryTable

Objects in this table store historical samples of Ethernet statistics for an Ethernet interface.

Supported objects:

Object Type	Syntax	Access	Functions
etherHistoryIndex	Integer32	read-only	None
etherHistorySampleIndex	Integer32	read-only	None
etherHistoryIntervalStart	TimeTicks	read-only	None
etherHistoryDropEvents	Counter32	read-only	None
etherHistoryOctets	Counter32	read-only	None
etherHistoryPkts	Counter32	read-only	None
etherHistoryBroadcastPkts	Counter32	read-only	None
etherHistoryMulticastPkts	Counter32	read-only	None
etherHistoryCRCAlignErrors	Counter32	read-only	None
etherHistoryUndersizePkts	Counter32	read-only	None
etherHistoryOversizePkts	Counter32	read-only	None
etherHistoryFragments	Counter32	read-only	None
etherHistoryJabbers	Counter32	read-only	None
etherHistoryCollisions	Counter32	read-only	None
etherHistoryUtilization	Integer32	read-only	None

alarmTable

Objects in this table store parameters that set up a periodic checking for alarm conditions.

Supported objects:

Object Type	Syntax	Access	Functions
alarmIndex	Integer32	read-only	None
alarmInterval	Integer32	read-create	rmon_set_alarm_interval
alarmVariable	OBJECT IDENTIFIER	read-create	rmon_set_alarm_variable
alarmSampleType	INTEGER	read-create	rmon_set_alarm_sample_type
alarmStartupAlarm	INTEGER	read-create	rmon_set_alarm_start_up
alarmRisingThreshold	Integer32	read-create	rmon_set_alarm_rising_threshold
alarmFallingThreshold	Integer32	read-create	rmon_set_alarm_falling_threshold
alarmRisingEventIndex	Integer32	read-create	rmon_set_alarm_rising_event_index

Object Type	Syntax	Access	Functions
alarmFallingEventIndex	Integer32	read-create	rmon_set_alarm_falling_event_index
alarmOwner	OwnerString	read-create	rmon_set_alarm_owner
alarmStatus	EntryStatus	read-create	rmon_snmp_set_alarm_status
alarmValue	Integer32	read-only	None

eventTable

Objects in this table store parameters that describe an event to generate when certain conditions are met.

Supported objects:

Object Type	Syntax	Access	Functions
eventIndex	Integer32	read-only	None
eventDescription	DisplayString	read-only	rmon_snmp_set_event_description
eventType	INTEGER	read-only	rmon_snmp_set_event_type
eventCommunity	OCTET STRING	read-only	rmon_snmp_set_event_community
eventLastTimeSent	TimeTicks	read-only	None
eventOwner	OwnerString	read-create	rmon_snmp_set_event_owner
eventStatus	EntryStatus	read-create	rmon_snmp_set_event_status

Unsupported objects:

- logEventIndex
- logIndex
- logTime
- logDescription

MIB API

This section describes functions related to reading and writing MIBs.

Function	Description
alarmTable	Queries an SNMP variable in the alarmTable
etherHistoryTable	Queries an SNMP variable in the etherHistoryTable
etherStatsTable	Queries an SNMP variable in the etherStatsTable
eventTable	Queries an SNMP variable in the eventTable

Function	Description
historyControlTable	Queries an SNMP variable in the historyControlTable
write_alarmTable	Sets an SNMP variable in the alarmTable
write_etherStatsTable	Sets an SNMP variable in the etherStatsTable
write_eventTable	Sets an SNMP variable in the eventTable
write_historyControlTable	Sets an SNMP variable in the historyControlTable

alarmTable

This function is called when an external network management system queries an SNMP variable in the alarmTable.

Syntax

```
u_int8_t *
alarmTable (struct variable *vp, oid *name, size_t *length, int exact,
size_t *var_len, WriteMethod **write_method, u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure
name	Pointer to the variable name (OID)
length	Number of elements (sub-ids) in the name
exact	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
vr_id	Unused

Output Parameters

var_len	Length of the variable that was read
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

etherHistoryTable

This function is called when an external network management system queries an SNMP variable in the etherHistoryTable.

Syntax

```
u_int8_t *
etherHistoryTable (struct variable *vp, oid *name, size_t *length, int exact, size_t
*var_len, WriteMethod **write_method, u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure

`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

etherStatsTable

This function is called when an external network management system queries an SNMP variable in the etherStatsTable.

Syntax

```
u_int8_t *
etherStatsTable (struct variable *vp, oid * name, size_t * length, int exact,
size_t * var_len, WriteMethod ** write_method, u_int32_t vr_id)
```

Input Parameters

`vp`	Pointer to SNMP variable structure
`name`	Pointer to the variable name (OID)
`length`	Number of elements (sub-ids) in the name
`exact`	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
`vr_id`	Unused

Output Parameters

`var_len`	Length of the variable that was read
`write_method`	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

eventTable

This function is called when an external network management system queries an SNMP variable in the eventTable.

Syntax

```
u_int8_t *
eventable (struct variable *vp, oid *name, size_t *length, int exact,
```

```
size_t *var_len, WriteMethod **write_method, u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure
name	Pointer to the variable name (OID)
length	Number of elements (sub-ids) in the name
exact	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
vr_id	Unused

Output Parameters

var_len	Length of the variable that was read
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

historyControlTable

This function is called when an external network management system queries an SNMP variable in the historyControlTable.

Syntax

```
u_int8_t *
historyControlTable (struct variable *vp, oid *name, size_t *length, int exact,
size_t *var_len, WriteMethod **write_method, u_int32_t vr_id)
```

Input Parameters

vp	Pointer to SNMP variable structure
name	Pointer to the variable name (OID)
length	Number of elements (sub-ids) in the name
exact	Whether this request is a GET (exact match: PAL_TRUE) or a GETNEXT (PAL_FALSE)
vr_id	Unused

Output Parameters

var_len	Length of the variable that was read
write_method	Pointer to a pointer to the SET function (WriteMethod) for the variable

Return Value

A pointer to the value of the variable when the function succeeds

Zero (0) or NULL when the function fails

write_alarmTable

This function is called when an external network management system sets an SNMP variable in the alarmTable.

Syntax

```
int
write_alarmTable (int action, u_char *var_val, u_char var_val_type, size_t var_val_len,
u_char *statP, oid *name, size_t length, struct variable *vp, u_int32_t vr_id)
```

Input Parameters

action	Unused
var_val	Value of the variable
var_val_type	Data type of the variable; one of these constants from lib/asn1.h which correspond to ASN.1's built-in simple types: ASN_INTEGER ASN_OCTET_STR ASN_OBJECT_ID
var_val_len	Length of the variable
statP	Unused
name	Pointer to variable name (OID)
length	Length of the name
vp	Pointer to SNMP variable structure
vr_id	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_BADVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

write_etherStatsTable

This function is called when an external network management system sets an SNMP variable in the etherStatsTable.

Syntax

```
int
write_etherStatsTable (int action, u_char *var_val, u_char var_val_type,
size_t var_val_len, u_char *statP, oid *name, size_t length, struct variable *vp,
u_int32_t vr_id)
```

Input Parameters

action	Unused
var_val	Value of the variable

`var_val_type`	Data type of the variable; one of these constants from lib/asn1.h which correspond to ASN.1's built-in simple types:
	`ASN_INTEGER`
	`ASN_OCTET_STR`
`var_val_len`	Length of the variable
`statP`	Unused
`name`	Pointer to variable name (OID)
`length`	Length of the name
`vp`	Pointer to SNMP variable structure
`vr_id`	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_BADVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

write_eventTable

This function is called when an external network management system sets an SNMP variable in the eventTable.

Syntax

```
int
write_eventTable (int action, u_char *var_val, u_char var_val_type, size_t var_val_len,
u_char *statP, oid *name, size_t length, struct variable *vp, u_int32_t vr_id)
```

Input Parameters

`action`	Unused
`var_val`	Value of the variable
`var_val_type`	Data type of the variable; one of these constants from lib/asn1.h which correspond to ASN.1's built-in simple types:
	`ASN_INTEGER`
	`ASN_OCTET_STR`
`var_val_len`	Length of the variable
`statP`	Unused
`name`	Pointer to variable name (OID)
`length`	Length of the name
`vp`	Pointer to SNMP variable structure

`vr_id`	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_BADVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

write_historyControlTable

This function is called when an external network management system sets an SNMP variable in the historyControlTable.

Syntax

```
int
write_historyControlTable (int action, u_char *var_val, u_char var_val_type,
size_t var_val_len, u_char *statP, oid *name, size_t length, struct variable *vp,
u_int32_t vr_id)
```

Input Parameters

`action`	Unused
`var_val`	Value of the variable

`var_val_type`	Data type of the variable; one of these constants from lib/asn1.h which correspond to ASN.1's built-in simple types:
	`ASN_INTEGER`
	`ASN_OCTET_STR`
	`ASN_OBJECT_ID`

`var_val_len`	Length of the variable
`statP`	Unused
`name`	Pointer to variable name (OID)
`length`	Length of the name
`vp`	Pointer to SNMP variable structure
`vr_id`	Unused

Output Parameters

None

Return Values

SNMP_ERR_NOERROR when the function succeeds

SNMP_ERR_GENERR when the function fails

SNMP_ERR_BADVALUE when var_val is invalid

SNMP_ERR_WRONGTYPE when var_val_type is invalid

SNMP_ERR_WRONGLENGTH when var_val_len is invalid

Index

M

mstp_api_add_instance 56
mstp_api_add_port 57, 58
mstp_api_delete_instance 58
mstp_api_disable_bridge 60
mstp_api_enable_bridge 60
mstp_api_get_bridge_l2gp_pathcost_control 110
mstp_api_get_loopguard_effective_status 62
mstp_api_get_loopguard_status 62
mstp_api_get_msti_port_path_cost 63
mstp_api_get_msti_port_path_cost_method 64
mstp_api_get_port_errdisable_timeout_interval 64
mstp_api_get_port_errdisable_timeout_status 65
mstp_api_get_port_forceversion 65
mstp_api_get_port_path_cost 66
mstp_api_get_port_path_cost_method 66
mstp_api_get_port_role 67

N

P

R

S

V

W

X