ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Shortest Path Bridging
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 Shortest Path Bridging (SPB).

Audience

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

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, *Shortest Path Bridging Overview*
- Chapter 2, *Shortest Path Bridging Command API*

Related Documents

The following documents are related to this document:

- *Shortest Path Bridging Command Reference*
- *Shortest Path Bridging Configuration Guide*

For more about Provider Backbone Bridging, see:

- *Carrier Ethernet Command Reference*
- *Carrier Ethernet Developer Guide*
- *Carrier Ethernet Configuration Guide*

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

Support

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

Comments

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

CHAPTER 1 Shortest Path Bridging Overview

This chapter is an overview of the ZebOS-XP implementation of Shortest Path Bridging.

IEEE 802.1aq Shortest Path Bridging (SPB) is a control plane protocol that combines an Ethernet data path with an IS-IS (Intermediate System To Intermediate System) link state protocol running between bridges. SPB does not depend on spanning tree protocols to provide a loop-free topology, but instead uses IS-IS link state packets (LSPs) to discover and advertise the network topology and compute the shortest path trees from all bridges in the SPB area. RFC 6329 describes the IS-IS extensions to support 802.1aq SPB.

There are two types of SPB depending on the type of Ethernet data path:

- Shortest Path Bridging - VID (SPBV) uses an 802.1ad “Q-in-Q” data path.
- Shortest Path Bridging - MAC (SPBM) uses an 802.1ah “MAC-in-MAC” data path.

SPBV and SPBM share a control plane, algorithms, and common routing mechanisms. Both SPBV and SPBM are fully interoperable with spanning tree technologies (MSTP and RSTP) at the SPB edge.

Shortest Path Bridging - VID

Each VLAN that is handled by SPBV uses an shortest path tree (SPT) set. An SPVID (Shortest Path VLAN Identifier) is assigned (manually or automatically) to each SPT in the set. This SPVID is registered by the ISIS-SPB control plane along a shortest path tree (SPT) rooted at the SPT Bridge to which it is assigned. The SPVID-to-SPT mapping information is sent to other bridges using ISIS-SPB.

The SPBV frame format is based on IEEE 802.1ad which is also called Provider Bridging or “Q-in-Q”. The frame format inserts an additional VLAN header into a single 802.1Q (Virtual LANs) Ethernet frame. There are two types of VLAN headers:

- The C-TAG (customer tag) or inner tag which is closest to the payload portion of the frame
- The S-TAG (service tag) or outer tag which is closest to the Ethernet header

At ingress into an SPBV region, the VID in a C-TAG or S-TAG of a customer frame is translated to the SPVID corresponding to the SPT that supports that VID. When customer frames do not contain a C-TAG or S-TAG, SPBV adds a tag with the SPVID.

At egress out of an SPBV region, the SPVID is translated back to the original VID and the C-TAG or S-TAG is removed.

SPBV uses shared learning among the set of SPVIDs that support a given SPBV VLAN. In an SPBV region, MAC addresses of end stations are learned at each bridge in the path.

The SPTs determined by ISIS-SPB provide symmetric bidirectional paths between any pair of SPT bridges within an SPT region. Symmetric means that the paths are the same in both directions. The same paths are used for both unicast and multicast traffic making them congruent. The congruency of the paths is essential to allow shared learning, where one filtering database is shared by all VLANs.

Shared learning is required because frames forwarded *from* a given SPT bridge contain a different SPVID than frames being forwarded *to* that SPT bridge. This permits a properly constructed mesh of shortest path trees constructed from unidirectional VLANs to use traditional flooding and learning outside a spanning tree context.

Existing MAC registration protocols for multicast groups can interoperate with SPBV and registrations received at the edge of an SPBV region are advertised throughout the region using IS-IS.

SPBV in a Customer Network

Figure 1-1 shows an SPT region in a customer network.

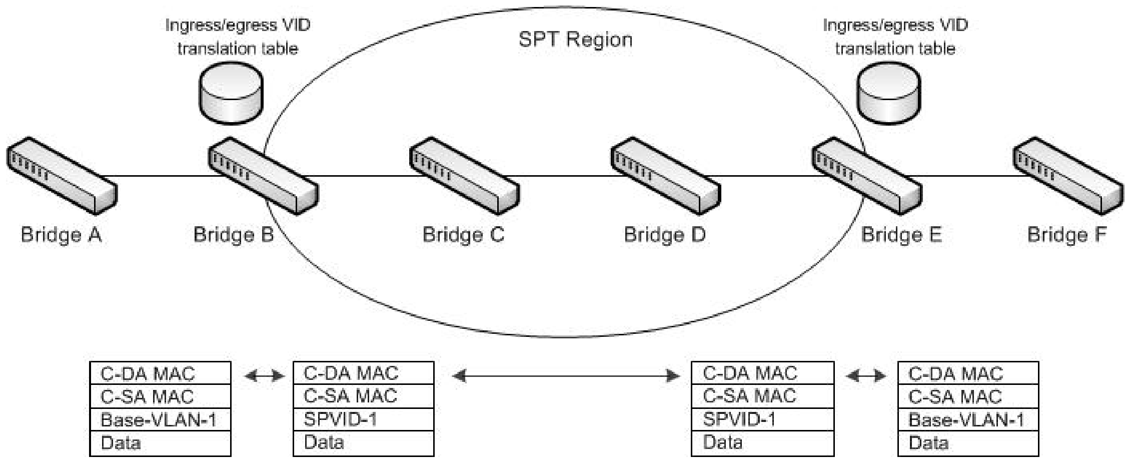

Figure 1-1: SPBV in a Customer Network

In Figure 1-1:

- Bridges A and F are compliant with 802.1Q and send and receive frames with a single VLAN identifier
- Bridges B-E are compliant with 802.1ad and have SPBV enabled.
- When bridge B receives a frame from bridge A, it translates the VLAN identifier (“Base-VLAN-1”) into a SPVID (“SPVID-1”). In the SPT region, forwarding is based on the SPVID.
- When bridge E receives a frame from the SPT region, it translates the SPVID into the VLAN identifier.

SPBV in a Provider Network

Figure 1-2 shows an SPT region in a provider network.

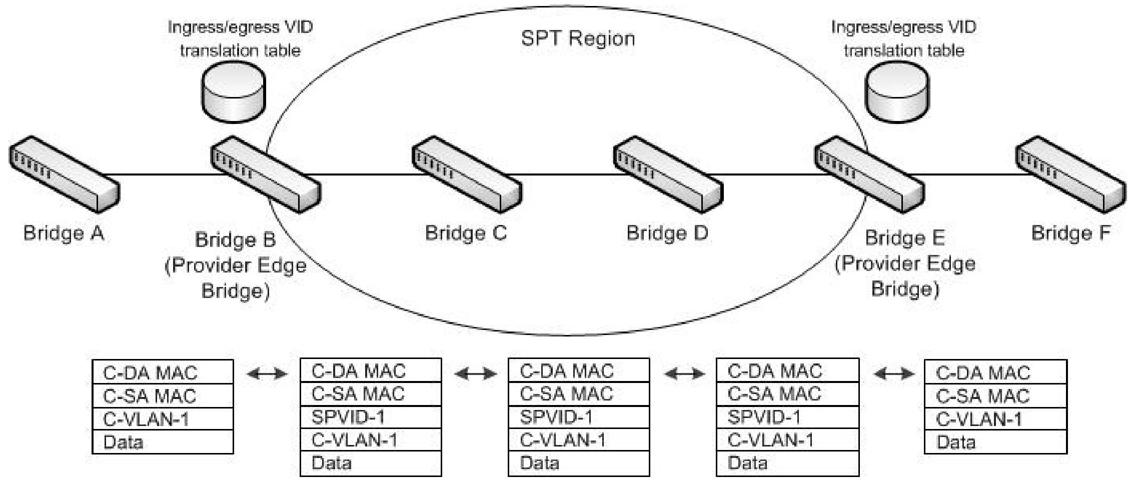

Figure 1-2: SPBV in a Provider Network

In Figure 1-2:

- Bridges A and F are compliant with 802.1Q.
- Bridges B-E are compliant with 802.1ad and have SPBV enabled.
- When bridge B receives a frame from bridge A, it adds an SPVID (“SPVID-1”). In the SPT region, forwarding is based on the SPVID.
- When bridge E receives a frame from the SPT region, it removes the SPVID

SPBV in a Provider Backbone Network

Figure 1-3 shows an SPT region in a provider backbone network.

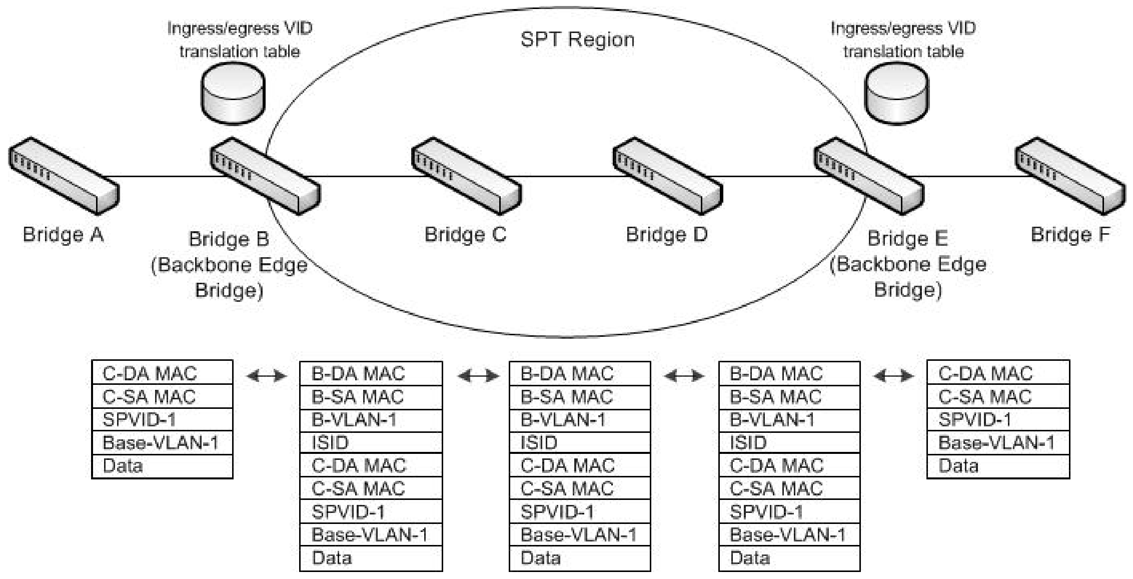

Figure 1-3: SPBV in a Provider Backbone Network

In Figure 1-3:

- Bridges A and F are compliant with 802.1Q and have SPBV enabled
- Bridges B-E are 802.1ah compliant and are *not* SPBV enabled.
- When bridge B receives a frame from bridge A, it adds backbone MAC addresses, a backbone VLAN identifier, and an ISID
- When bridge E receives a frame from the SPT region, it removes the backbone MAC addresses, backbone VLAN identifier, and an ISID

Shortest Path Bridging - MAC

SPBM supports a VLAN by using one or more Backbone MAC addresses to identify each node and its associated SPT, and it can support multiple forwarding topologies for load sharing across equal cost trees using a single B-VID per forwarding topology.

In SPBM, the backbone MAC addresses of the participating nodes and the service membership information for interfaces to UNI (User Network Interface) ports are distributed. A calculation engine uses topology data to compute symmetric shortest path trees based on minimum cost from each participating node to all other participating nodes. The shortest path trees are then used to populate forwarding tables for each participating node's individual MAC addresses and for group addresses. Group multicast trees are sub trees of the default shortest path tree formed by (source, group) pairing.

The SPBM frame format is based on IEEE 802.1ah which is also called Provider Backbone Bridging (PBB) or “MAC-in-MAC”. It is called the latter because it involves encapsulating backbone source and destination addresses (termed B-SA and B-DA) along with the data and customer MAC addresses (C-DA and C-SA) in an frame. Sometimes, this is referred to as hiding or encapsulating customer MAC addresses within backbone MAC addresses.

Note: The combination of B-SA and B-DA is sometimes called “B-MAC”; the combination of C-SA and C-DA is sometimes call “C-MAC”.

The “hiding/encapsulating” of customer MAC addresses in backbone MAC addresses means that the backbone does not need to learn customer MAC addresses. Customer MAC addresses are learned at BEB ports only.

The IEEE 802.1ah header includes a service instance identifier (I-SID) which is a label that maps to a customer VLAN ID. An I-SID virtualizes VLANs across a network. VLANs are mapped into I-SIDs by configuring only the edge of the network at the Backbone Edge Bridges (BEBs).

In the SPBM core, the bridges are referred to as Backbone Core Bridges (BCBs). BCBs forward encapsulated traffic based on the B-DA

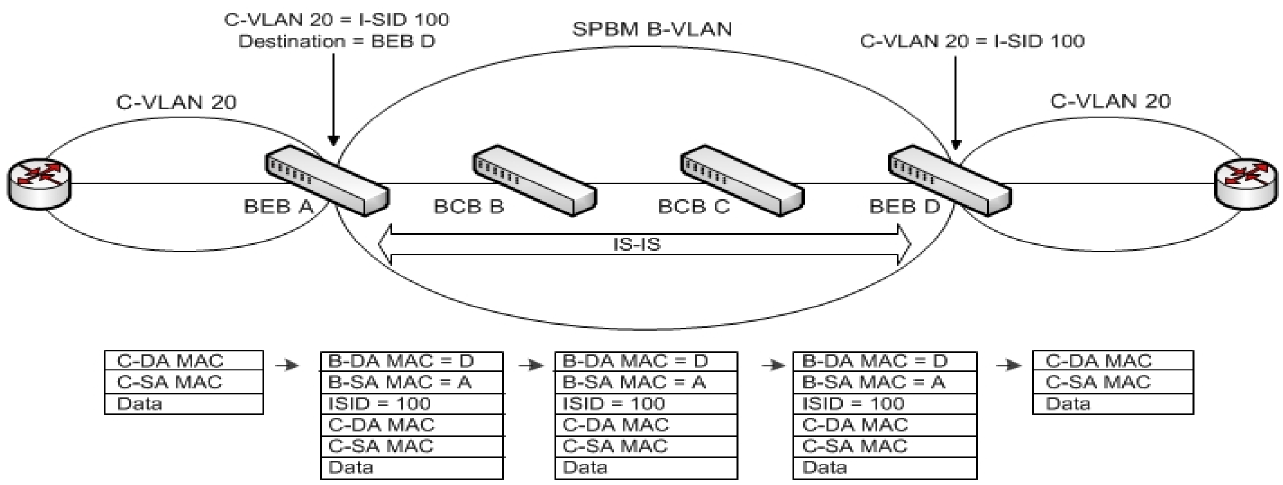

Figure 1-4: SPBM Network Topology

Figure 1-4 shows a basic SPBM network topology. Switches A and D are BEBs that provide the boundary between the customer VLAN (C-VLAN) and the backbone VLAN (B-VLAN). Switches B and C are the BCBs that form the core of the SPBM network.

In this example, BEB A and BEB D are configured to associate C-VLAN 20 with I-SID 100.

1. When BEB A receives traffic from C-VLAN 20 that must be forwarded to the far-end location, it performs a lookup and determines that C-VLAN 20 is associated with I-SID 100 and that BEB D is the destination for I-SID 100.

2. BEB A then encapsulates the data and customer MAC addresses (C-DA and C-SA) into a new frame, using its own node's address (A) as the B-SA and D as the B-DA. BEB A then forwards the encapsulated traffic to BCB B.
3. To forward traffic in the core toward the destination node D, BCB B and BCB C perform Ethernet switching using the B-DA only.
4. At BEB D, the node strips the B-SA and B-DA and performs a lookup to determine the destination for traffic with I-SID 100. BEB D identifies the destination as C-VLAN 20 and forwards the frame to the appropriate VLAN and port.

Note: Depending on the topology, several different equal cost multi path trees are possible and SPB supports multiple algorithms per IS-IS instance. In ZebOS-XP, only one ECT (equal cost tree) and one IS-IS instance are supported.

CHAPTER 2 Shortest Path Bridging Command API

This chapter contains the Application Programming Interface (API) for the Shortest Path Bridging (SPB) commands.

Data Structures

ipi_vr

See the *Virtual Routing Developer Guide* for a description of this structure.

spb_bridge

This data structure defined in `spbd/spb_types.h` holds bridge-related information for each SPB bridge.

Member	Description
`bridge_name`	Bridge name
`bridge_addr`	Bridge MAC address

`bridge_type`	Bridge type:
	`SPB_BRIDGE_TYPE_BCB`
	`SPB_BRIDGE_TYPE_BEB_ICOMP`
	`SPB_BRIDGE_TYPE_BEB_BCOMP`
	`SPB_BRIDGE_TYPE_SPBV_CVLAN_EDGE`
	`SPB_BRIDGE_TYPE_SPBV_CVLAN`
	`SPB_BRIDGE_TYPE_SPBV_SVLAN_EDGE`
	`SPB_BRIDGE_TYPE_SPBV_SVLAN_CORE`
	`SPB_BRIDGE_TYPE_SPBV_BEB`
	`SPB_BRIDGE_TYPE_SPBV_BCB`
	`SPB_BRIDGE_TYPE_EDGE` (Both SPBM and SPBV)
	`SPB_BRIDGE_TYPE_CORE` (Both SPBM and SPBV)

`path_cost_method`	Used for Path cost method short/long
`learning_enabled`	Flag representing learning state
`is_enabled`	Bridge enable flag
`mcid`	MST configuration identifier
`aux_mcid`	Auxiliary MST configuration identifier
`digest`	Agreement digest
`spb_mtid_list`	Multi-topology related information
`port_list`	Ports

Member	Description
`low_port`	Index of the lowest numbered port (by ifindex)
`spb_bvlan_list`	VLAN-related information
`cist_bvlan_bmp`	VLAN bitmap
`top`	Pointer to ISIS-SPB (of type `spbi`)
`cist_bridge_id`	CIST bridge identifier (of type `bridge_id`)
`cist_root_bridge_id`	CIST root bridge identifier (of type `bridge_id`)
`cist_bridge_priority`	Bridge priority
`cist_external_rpc`	External root path cost
`spb_bridge_instance_list`	SPB instances
`spb_recv_thread`	SPB read thread
`t_mt_tlv`	TLV 144 update thread
`ipvpn`	IPVPN enable flag
`ipvpn_nbr_list`	Holds neighbor information (of type `ipvpn_nbr_node`)
`t_self_update`	IPVPN thread
`t_ipvpn_tlv`	IPVPN TLV thread
`config_flags`	Flag to represent various configuration settings

Definition

```
struct spb_bridge
{
  u_int8_t                        bridge_name[L2_BRIDGE_NAME_LEN+1];
  u_char                          bridge_addr[ETHER_ADDR_LEN];
  u_int8_t                        bridge_type;

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

  u_int8_t                        is_enabled;

  struct spb_config_info          mcid;
  struct spb_config_info          aux_mcid;

  struct spb_agreement_digest     digest;

  struct list                     * spb_mtid_list;  /* spb_mtid */
```

```
/*--------------------------------------------
PORT
--------------------------------------------*/
  struct list                     * port_list;

/* Index of the lowest numbered port (by ifindex) */
  u_int32_t                       low_port;

/*--------------------------------------------
VLAN
--------------------------------------------*/
  struct avl_tree                 * spb_bvlan_list; /* spb_bvlan */
  struct spb_vlan_bmp             cist_bvlan_bmp;

/*--------------------------------------------
ISIS-SPB
--------------------------------------------*/
  struct spbi                     * top;

/*--------------------------------------------
CIST
--------------------------------------------*/
  struct bridge_id                cist_bridge_id;
  struct bridge_id                cist_root_bridge_id;
  u_int32_t                       cist_bridge_priority;
  u_int32_t                       cist_external_rpc;

/*--------------------------------------------
SPB Instances
--------------------------------------------*/
  struct list                     * spb_bridge_instance_list;

/*--------------------------------------------
Threads
--------------------------------------------*/
  struct thread                   * spb_recv_thread;
  struct thread                   * t_mt_tlv;

#ifdef HAVE_IPVPN_SPB
/*--------------------------------------------
 IPVPN
--------------------------------------------*/
  u_int8_t                        ipvpn;
#define SPB_IPVPN_ENABLE          (1 << 0)

 struct avl_tree                 * ipvpn_nbr_list;

  struct thread                   * t_self_update;
  struct thread                   * t_ipvpn_tlv;
#endif /* HAVE_IPVPN_SPB */
```

```
/*--------------------------------------------
 LOOP MITIGATION & PREVENTION
--------------------------------------------*/
  u_int8_t                        config_flags;
#define SPB_LOOP_MITIGATION_ENABLE          (1 << 0)
#define SPB_LOOP_PREVENTION_ENABLE          (1 << 1)
#define SPB_LSP_BROADCAST_ENABLE            (1 << 2)
};
```

spb_port

This data structure defined in `spbd/spb_types.h` holds port related information when a port is mapped to an SPB bridge.

Member	**Description**
`br`	Bridge information (of type `spb_bridge`)
`port_instance_list`	Port instance information
`mac_addr`	MAC address
`if_index`	Interface address
`port_name`	Interface name

`port_type`	Port type:
	`SPB_VLAN_PORT_MODE_INVALID`
	`SPB_VLAN_PORT_MODE_CNP`
	`SPB_VLAN_PORT_MODE_VIP`
	`SPB_VLAN_PORT_MODE_PNP`
	`SPB_VLAN_PORT_MODE_CBP`
	`SPB_VLAN_PORT_MODE_PIP`

`is_enabled`	Enable/disable flag
`port_vlan_bmp`	VLAN bitmap
`is_isis_enabled`	Flag representing ISIS-SPB enable
`cist_path_cost`	CIST path cost
`cist_port_priority`	CIST priority
`cist_port_id`	CIST port identifier

`cist_port_state`	CIST state:
	`STATE_DISCARDING`
	`STATE_LISTENING`
	`STATE_LEARNING`
	`STATE_FORWARDING`
	`STATE_BLOCKING`

`all_spt_agree`	All SPT agree flag

Definition

```
struct spb_port
{
  struct spb_bridge                * br;
  struct list                      * port_instance_list;

  u_char                           mac_addr[ETHER_ADDR_LEN];
  u_int32_t                        if_index;
  u_char                           port_name[L2_IF_NAME_LEN];

  u_int8_t                         port_type;
  u_int8_t                         is_enabled;

  struct spb_vlan_bmp              port_vlan_bmp;

/*---------------------------------------------
ISIS-SPB
----------------------------------------------*/
  u_int8_t                         is_isis_enabled;

/*---------------------------------------------
CIST
----------------------------------------------*/
  u_int32_t                        cist_path_cost;

  u_int16_t                        cist_port_priority;
  u_int16_t                        cist_port_id;
  enum port_state                  cist_port_state;

  u_int8_t                         all_spt_agree:1;
};
```

Functions

The functions in this section implement the commands in the *Shortest Path Bridging Command Reference*.

Function	Description
rib_ip_vrf_isid_set	Creates a VRF instance associated with an I-SID
rib_ip_vrf_unset	Deletes a VRF instance
spb_api_disable_bridge	Disables SPB on a given bridge
spb_api_enable_bridge	Enables SPB on a given bridge
spb_api_find_bridge	Finds a given bridge
spb_api_map_bvlan_to_bridge_instance	Maps BVLANs to an SPB instance

Function	Description
spb_api_set_bridge_priority	Sets the bridge priority
spb_api_set_gmac	Sets a group MAC address for a BVLAN
spb_api_set_hostname	Sets the name of the host
spb_api_set_if_hello_interval	Sets an interface's hello interval
spb_api_set_if_hello_multiplier	Sets an interface's hello-multiplier value
spb_api_set_if_lsp_interval	Sets an interface's LSP transmission interval
spb_api_set_if_minimal_hello_interval	Sets the hold time in hello PDUs to 1 second
spb_api_set_if_retransmit_interval	Sets the LSP retransmission interval
spb_api_set_ignore_lsp_errors	Tells SPB to ignore LSPs with checksum errors
spb_api_set_ipvpn	Enables or disables IPVPN for SPB
spb_api_set_lsp_gen_interval	Sets the minimum interval between regenerating the same LSP
spb_api_set_lsp_max_lifetime	Sets the maximum LSP lifetime
spb_api_set_lsp_refresh_interval	Sets the LSP refresh interval
spb_api_set_overload_bit_options	Sets overload bit options
spb_api_set_port_instance_path_cost	Sets the path cost for a port in SPB
spb_api_set_spbv_mode	Sets the SPVID allocation mode
spb_api_set_spf_interval	Sets the minimum and maximum exponential backoff delay between the receipt of a topology change and calculating SPF
spb_api_set_spsourceid	Sets the shortest path source identifier
spb_api_set_spvid_pool	Sets the SPVID pool range
spb_api_set_system_id	Sets the system identifier for the ISIS-SPB process
spb_api_spvid_config	Maps a SPVID to a VID
spb_api_spvid_unconfig	Unmaps a SPVID from a VID
spb_api_unmap_bvlan_to_bridge_instance	Unmaps BVLANs from an SPB instance
spb_api_unset_gmac	Removes a group MAC address for a BVLAN
spb_api_unset_hostname	Deletes the name of the host
spb_api_unset_if_hello_interval	Sets the interface's hello interval to its default
spb_api_unset_if_hello_multiplier	Sets an interface's hello-multiplier value to its default
spb_api_unset_if_lsp_interval	Sets an interface's LSP transmission interval to its default
spb_api_unset_if_retransmit_interval	Sets the LSP retransmission interval to its default

Function	Description
spb_api_unset_ignore_lsp_errors	Tells SPB to validate LSP checksums and reject an LSP if it has a checksum error
spb_api_unset_lsp_gen_interval	Sets the minimum interval between regenerating the same LSP to its default
spb_api_unset_lsp_max_lifetime	Sets the maximum LSP lifetime to its default
spb_api_unset_lsp_refresh_interval	Sets the LSP refresh interval to its default
spb_api_unset_overload_bit	Clears the overload bit of self-LSPs
spb_api_unset_spf_interval	Sets the minimum and maximum exponential backoff delay between the receipt of a topology change and the calculation of the SPF to their default
spb_api_unset_spsourceid	Sets the shortest path source identifier to its default value
spb_api_unset_spvid_pool	Sets the SPVID pool range to its default
spb_api_unset_system_id	Sets the identifier for the ISIS-SPB process to NULL
spb_cist_disable_port	Disables SPB on an interface
spb_cist_enable_port	Enables SPB on an interface
spb_nsm_send_convention_id	Sets the agreement protocol convention
spb_nsm_send_loop_mitign	Enables or disables loop mitigation
spb_nsm_send_loop_prevention	Enables or disables loop prevention
spbi_api_del_mtid	Deletes a multi-topology identifier
spbi_api_set_mtid	Sets a multi-topology identifier

Include Files

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

- `nsm/rib/nsm_rib_vrf.h`
- `spbd/isis-spb/spb_isis_api.h`
- `spbd/spb_api.h`
- `spbd/spb_nsm.h`
- `spbd/spb_port.h`

rib_ip_vrf_isid_set

This function creates a VRF (Virtual Routing and Forwarding) instance associated with an I-SID (service instance identifier) that needs to advertise its routes over an SPB network.

This function implements the `ip vrf WORD isid` command.

Syntax

```
#include "ribd/rib_api.h"
int
```

```
rib_ip_vrf_isid_set (struct ipi_vr *vr, char *name, u_int32_t isid)
```

Input Parameters

`vr`	VRF instance
`name`	VRF name
`isid`	Service instance identifier

Output Parameters

None

Return Value

`RIB_API_SET_ERR_VRF_NAME_TOO_LONG` when `name` is more than 64 characters

`RIB_API_VRF_ISID_ALREADY_MAPPED` when `name` is already mapped to an I-SID

`RIB_API_SET_ERR_VRF_CANT_CREATE` when a system resource limit is exceeded

`RIB_API_SET_SUCCESS` when the function succeeds

rib_ip_vrf_unset

This function deletes a VRF (Virtual Routing and Forwarding) instance.

This function implements the `no ip vrf WORD` command.

Syntax

```
#include "ribd/rib_api.h"
int
rib_ip_vrf_unset (struct ipi_vr *vr, char *name)
```

Input Parameters

`vr`	VRF instance
`name`	VRF name

Output Parameters

None

Return Value

`RIB_API_SET_ERR_VRF_NOT_EXIST` when `name` is not not an VRF instance

`RIB_API_SET_SUCCESS` when the function succeeds

spb_api_disable_bridge

This function disables SPB on a given bridge.

This function implements the `bridge (<1-32> | backbone) spb disable` command.

Syntax

```
#include "spbd/spb_api.h"
int
```

```
spb_api_disable_bridge (char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NAME_IS_NULL` when `bridge_name` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`RESULT_ERROR` when the function fails

Less than zero when the function fails

Greater than zero when the function succeeds

spb_api_enable_bridge

This function enables SPB on a given bridge.

This function implements the `bridge (<1-32> | backbone) spb enable` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_enable_bridge (char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NAME_IS_NULL` when `bridge_name` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`RESULT_ERROR` when the function fails

Less than zero when the function fails

Greater than zero when the function succeeds

spb_api_find_bridge

This function returns a pointer to a given bridge.

This function is called by several different commands.

Syntax

```
#include "spbd/spb_api.h"
struct spb_bridge *
spb_api_find_bridge (char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

A pointer to the given bridge when the function succeeds

NULL when the bridge is not found

spb_api_map_bvlan_to_bridge_instance

This function maps BVLANs (Backbone Virtual Local Area Networks) to an SPB bridge instance.

This function implements the `bridge (<1-32> | backbone) instance (spbm|spbv) vlan` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_map_bvlan_to_bridge_instance ( char *bridge_name, spb_vid_t start,
                                       spb_vid_t end, u_int32_t instance_id,
                                       s_int16_t mt_id, s_int8_t ect,
                                       bool_t is_mstp))
```

Input Parameters

`bridge_name`	Bridge name
`start`	Starting backbone VLAN identifier
`end`	Ending backbone VLAN identifier
`instance_id`	Instance identifier; one of these constants from `spbd/spb_types.h`:
`SPB_SPBM_INSTANCE_ID`	Shortest Path Bridging MAC
`SPB_SPBV_INSTANCE_ID`	Shortest Path Bridging VID
`ect`	Equal-cost tree algorithm identifier:
`1`	Low path ID: The selected path includes the bridge with the numerically lowest bridge identifier. When the bridge priority value is equal for two bridge identifiers, the lower bridge identifier determines the priority (0,1,2,3, ...).
`2`	High path ID: The selected path includes the bridge with the numerically lowest bridge identifier after masking 0xFF which reverses the bridge priority values. When the bridge priority value is equal for two bridge identifiers, the lower bridge identifier determines the priority (15,14,13, ...).

`mt_id`	Multi-topology identifier <3996-4095>
`is_mstp`	Whether this bridge is part of an MSTP

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `start` is NULL

`RESULT_ERROR` when the bridge is not configured or when memory allocation fails

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

`SPB_API_ERR_MAX_VLANS_CONFIGURED` when the number of VLANs exceeds the maximum that can be configured

`SPB_API_ERR_VLAN_NOT_MAPPED_TO_BRIDGE` when a given VLAN is not mapped to a bridge

`SPB_API_ERR_MTID_NOT_FOUND` when `mt_id` is NULL or the default MTID is not found

`SPB_API_ERR_VLAN_NOT_MAPPED_TO_INST` when a given VLAN is not mapped to the bridge instance

`SPB_API_ERR_ISID_ARE_MAPPED_TO_BVLAN` when an ISID is mapped to a VLAN

`SPB_API_ERR_VLAN_NOT_MAPPED_TO_MTID` when the VLAN is not mapped to the given `mt_id`

`RESULT_OK` when the function succeeds

spb_api_set_bridge_priority

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

This function implements the `bridge (<1-32> | backbone) priority` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_bridge_priority ( char *bridge_name,
                              u_int32_t new_priority)
```

Input Parameters

`bridge_name`	Bridge name
`new_priority`	Bridge priority in increments of 4096 <0-61440>; a lower priority increases the chance of the bridge becoming root

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`RESULT_ERROR` when the bridge is not configured or when the function fails

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_PRIORITY_VALUE_WRONG_2` when `new_priority` is not a multiple of 4096

`SPB_API_ERR_PRIORITY_OUTOFBOUNDS` when `new_priority` is outside of the range

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`RESULT_OK` when the function succeeds

spb_api_set_gmac

This function sets a group MAC address for a base VLAN.

This function implements the `bridge (<1-32> | backbone) spbv bvlan <1-4094> group-mac` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_gmac(char *if_name, char *bridge_name, u_char *gmac_addr, spb_vid_t vid,
enum spb_gmac_mode mode, u_int8_t sr_bit)
```

Input Parameters

`if_name`	Interface name
`bridge_name`	Bridge name
`gmac_addr`	Group MAC address in `HHHH.HHHH.HHHH` format
`vid`	Base VLAN identifier <1-4094>
`mode`	Whether to receive and/or transmit; one of these constants from the `spb_gmac_mode` enum in `spbd/spb_types.h`:
`SPBV_GMAC_RX`	Receive only
`SPBV_GMAC_TX`	Transmit only
`SPBV_GMAC_TXRX`	Both receive and transmit
`sr_bit`	Service Request value:
`0`	Not declared
`1`	Forward all groups
`2`	Forward all unregistered groups

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`RESULT_ERROR` when:

- The backbone VLAN cannot be found
- Memory allocation fails

`RESULT_OK` when the function succeeds

spb_api_set_hostname

This function sets the name of the host for the SPB bridge.

This function implements the `isis-spb hostname` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_hostname (char *bridge_name, char *host_name)
```

Input Parameters

`bridge_name`	Bridge name
`host_name`	The name of the host.

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `host_name` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_ISIS_API_SET_ERROR` when the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the instance does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_if_hello_interval

This function sets an interface's hello interval.

This function implements the `isis-spb hello-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_if_hello_interval ( char *if_name,
                                u_int32_t interval,
                                int level)
```

Input Parameters

`if_name`	Interface name

`interval`	The interval in seconds <1-65535>
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_PORT_NOT_FOUND` when the bridge port is not created

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_INVALID_VALUE` when the given interval is outside of the range (1-65535)

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the interface parameter is not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_if_hello_multiplier

This function sets an interface's hello-multiplier value. The hello holding time is calculated by multiplying the hello interval by this value.

This function implements the `isis-spb hello-multiplier` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_if_hello_multiplier ( char *if_name, u_int32_t multi, int level)
```

Input Parameters

`if_name`	Interface name
`multi`	Multiplier for the hello holding time <2-100>
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_PORT_NOT_FOUND` when the bridge port is not created

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_INVALID_VALUE` when the given multiple is outside of the range (2-100)

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the interface parameter is not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_if_lsp_interval

This function sets an interface's LSP transmission interval.

This function implements the `isis-spb lsp-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_if_lsp_interval ( char *if_name, u_int32_t interval)
```

Input Parameters

`if_name`	Interface name
`interval`	The interval in milliseconds <1-4294967295>

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_PORT_NOT_FOUND` when the bridge port is not created

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the interface parameter is not configured

`SPB_API_ERR_INVALID_VALUE` when the given interval is outside of the range (1-4294967295)

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_if_minimal_hello_interval

This function sets the hold time in hello PDUs to 1 second.

This function implements the `isis-spb hello-interval minimal` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_if_minimal_hello_interval (char *if_name, int level)
```

Input Parameters

`if_name`	Interface name
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_PORT_NOT_FOUND` when the bridge port is not created

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the interface parameter is not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_if_retransmit_interval

This function sets the LSP retransmission interval.

This function implements the `isis-spb retransmit-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_if_retransmit_interval ( char *if_name, u_int32_t interval)
```

Input Parameters

`if_name`	Interface name
`interval`	The interval in milliseconds <0-65535>

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_PORT_NOT_FOUND` when the bridge port is not created

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_INVALID_VALUE` when the given interval is outside of the range (1-65535)

`SPB_API_ERR_ISIS_IF_NOT_FOUND` when the global ISIS data structure is NULL

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the interface parameter is not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_ignore_lsp_errors

This function tells SPB to ignore LSPs with checksum errors. By default, ZebOS-XP validates the checksum when it receives an LSP and if there is an error, the LSP is dropped.

This function implements the `isis-spb ignore-lsp-errors` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_ignore_lsp_errors (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance area tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance area tag does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_ipvpn

This function enables or disables IPVPN for SPB. When IPVPN is enabled, a Virtual Route Forwarder (VRF) is identified by an I-SID (service instance identifier). IPVPN traffic within the SPB network uses the I-SID portion of the Service Instance TAG (I-TAG) without a C-MAC header, called the short I-TAG.

This function implements the `ipvpn enable` and `ipvpn disable` commands.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_ipvpn (u_int8_t enable)
```

Input Parameters

`enable`	Whether to enable or disable IPVPN:
`PAL_TRUE`	Enable IPVPN.
`PAL_FALSE`	Disable IPVPN.

Output Parameters

None

Return Value

`RESULT_ERROR` when the backbone bridge is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the backbone bridge is NULL but the port instance configuration exists

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the backbone bridge is not found

`SPB_API_ERR_IPVPN_INIT_FAILED` when the IPVPN initialization fails

`RESULT_OK` when the function succeeds

spb_api_set_lsp_gen_interval

This function sets the minimum interval between regenerating the same LSP.

This function implements the `isis-spb lsp-gen-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_lsp_gen_interval ( char *bridge_name, char *tag,
                               int level, u_char interval)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance area tag
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2
`interval`	The interval in seconds <1-120>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_API_ERR_INVALID_VALUE` when the given interval is outside of the range (1-120)

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_lsp_max_lifetime

This function sets the maximum LSP lifetime.

This function implements the `isis-spb max-lsp-lifetime` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_lsp_max_lifetime ( char *bridge_name, char *tag,
                               u_int32_t max_lifetime)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag
`max_lifetime`	Maximum LSP lifetime in seconds <350-65535>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the port instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_API_ERR_INVALID_VALUE` when `max_lifetime` is outside of the range (350-65535)

`SPB_ISIS_API_SET_ERR_LT_REFRESH_TIME` when the `max_lifetime` is less than or equal the LSP refresh interval

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_lsp_refresh_interval

This function sets the LSP refresh interval.

This function implements the `isis-spb lsp-refresh-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_lsp_refresh_interval (char *bridge_name, char *tag,
                                  u_int32_t interval)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag
`interval`	The interval in seconds <1-65535>

Output Parameters

None.

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the bridge instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_API_ERR_INVALID_VALUE` when the given interval is outside of the range <1-65535>

`SPB_ISIS_API_SET_ERR_GT_MAX_LSP_LIFETIME` when the given interval is greater than the maximum LSP lifetime

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_overload_bit_options

This function sets overload bit options.

This function implements the `isis-spb set-overload-bit` command.

Note: The `startup_val` and `wait_for_bgp` parameters are mutually exclusive. Do not set both set to `PAL_TRUE`.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_overload_bit_options (char *bridge_name, s_int16_t mt_id,
                                  bool_t args, bool_t startup_val,
                                  int interval,
                                  bool_t wait_for_bgp,
                                  bool_t suppress_external,
                                  bool_t suppress_interlevel)
```

Input Parameters

`bridge_name`	Bridge name
`mt_id`	Multi-topology identifier <3996-4095>

`args`	Automatically or manually enable options:
`PAL_FALSE`	Always enable `startup_val`, `suppress_external`, and `suppress_interlevel`
`PAL_TRUE`	Check each of these parameters and enable them as specified in this function call
`startup_val`	Whether to set the overload bit after a restart:
`PAL_TRUE`	Set the overload-bit temporarily after a reboot
`PAL_FALSE`	Do not set the overload-bit temporarily after a reboot
`interval`	Time in seconds to advertise as overloaded after a reboot
`wait_for_bgp`	Whether BGP should decide to unset the overload bit:
`PAL_TRUE`	Let BGP decide when to unset the overload bit
`PAL_FALSE`	Do not let BGP decide when to unset the overload bit
`suppress_external`	Whether to suppress IP prefixes learned from other protocols:
`PAL_TRUE`	Do not advertise IP prefixes learned from other protocols
`PAL_FALSE`	Advertise IP prefixes learned from other protocols
`suppress_interlevel`	Whether to suppress IP prefixes learned from another ISIS level:
`PAL_TRUE`	Do not advertise IP prefixes learned from another ISIS level
`PAL_FALSE`	Advertise IP prefixes learned from another ISIS level

Output Parameters

None

Return Values

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`SPB_ISIS_API_SET_ERROR` when the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge instance configuration exists

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_SET_ERR_OVERLOAD_OPTION_INVALID` when the combination of parameters is not valid

`RESULT_ERROR` when there is some other kind of error

`RESULT_OK` when the function succeeds

spb_api_set_port_instance_path_cost

This function sets the path cost for an interface.

This function implements the `bridge-group (<1-32> | backbone) spb path-cost` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_port_instance_path_cost ( char *bridge_name, char *if_name,
                                      s_int16_t mt_id, u_int32_t cost)
```

Input Parameters

`bridge_name`	Bridge name
`if_name`	Interface name
`mt_id`	Multi-topology identifier <3996-4095>
`cost`	Path cost <1-16777215>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is not found

`RESULT_ERROR` when the bridge is not configured or the port instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is not found

`SPB_API_ERR_PATHCOST_OUTOFBOUNDS` when `cost` is out of range <1-16777215>

`SPB_API_ERR_PORT_NOT_FOUND` when the port is not found

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

`SPB_API_ERR_PORT_INSTANCE_NOT_FOUND` when the port instance is NULL

`SPB_API_ERR_MTID_NOT_FOUND` when `mt_id` is NULL

`RESULT_OK` when the function succeeds

spb_api_set_spbv_mode

This function sets the Shortest Path VLAN identifier (SPVID) allocation mode.

This function implements the `bridge (<1-32> | backbone) spbv mode (auto|manual)` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_spbv_mode(char *bridge_name, u_int8_t mode)
```

Input Parameters

`bridge_name`	Bridge name
`mode`	Whether to allocate SPVIDs automatically or manually; one of these constants from `spbd/spb_types.h`:

`SPBV_AUTO_MODE`

Automatically allocate SPVIDs (default setting).

`SPBV_MANUAL_MODE`

Manually allocate SPVIDs.

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`RESULT_ERROR` when the bridge instance configuration is NULL

`RESULT_OK` when the function succeeds or when the allocation is already set to `mode`

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

spb_api_set_spf_interval

This function sets the minimum and maximum exponential backoff delay between receiving a topology change and calculating the Shortest Path First (SPF).

This function implements the `isis-spb spf-interval-exp` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_spf_interval (char *bridge_name, char *tag, int level,
                          u_int32_t min_delay, u_int32_t max_delay)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance area tag
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2
`min_delay`	Minimum delay in milliseconds <0-2147483647>
`max_delay`	Maximum delay in milliseconds <0-2147483647>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the bridge instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_INVALID_VALUE` when `min_delay` or `max_delay` is out of range <0-2147483647>

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_set_spsourceid

This function sets the shortest path source identifier. This identifier must be unique within the area.

This function implements the `bridge (<1-32> | backbone) spsourceid` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_spsourceid ( char *bridge_name,
                         u_int32_t spsourceid)
```

Input Parameters

`bridge_name`	Bridge name
`spsourceid`	The shortest path source identifier <1-1048575>. Specify `SPB_BRIDGE_AUTO_SPSOURCEID` in `spbd/spb_api.h` to generate the shortest path source identifier automatically.

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NAME_IS_NULL` when `bridge_name` is NULL

`RESULT_ERROR` when the bridge is NULL and the bridge configuration is not present

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_SPSOURCEID_OVERWRITE` when the source identifier is already set

`RESULT_OK` when the function succeeds

spb_api_set_spvid_pool

This function sets the Shortest Path VLAN identifier (SPVID) pool range.

The default SPVID pool range is <3600-3999>.

This function implements the `bridge (<1-32> | backbone) spvid-pool <1-4094> to <1-4094>` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_set_spvid_pool(char *bridge_name, u_int32_t start, u_int32_t last)
```

Input Parameters

`bridge_name`	Bridge name
`start`	Starting SPVID <1-4094>
`last`	Ending SPVID <1-4094>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`RESULT_ERROR` when:

- The bridge instance configuration is NULL
- The bridge is not configured for SPB or SPBV
- `start` is greater than the current starting SPVID and a SPVID less than `start` is already being used
- `last` is less than the current ending SPVID and a SPVID greater than `last` is already being used

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

`RESULT_OK` when the function succeeds

spb_api_set_system_id

This function sets the network-wide unique identifier for the ISIS-SPB process.

This function implements the `isis-spb system-id` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_set_system_id (char *bridge_name, char *tag, u_char *system_id)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance area tag
`system_id`	System identifier in `XX.XX.XX.XX.XX.XX` format with 6 hexadecimal numbers separated by periods

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when a parameter is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL and the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the ISIS master is NULL but the bridge instance configuration exists

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`RESULT_ERROR` when the `system_id` is NULL

`SPB_ISIS_SET_ERR_NET_WRONG_FORMAT` when the `system_id` is NULL or is an empty string or does not contain hexadecimal digits and periods

`SPB_ISIS_SET_ERR_NET_INVALID_LENGTH` when the `system_id` exceeds the maximum length of 6 hexadecimal numbers

`SPB_ISIS_SET_ERR_SYSTEM_ID_CANT_CHANGED` when a different `system_id` is already set for the instance

`SPB_ISIS_SET_ERR_TOO_MANY_AREA_ADDRESSES` when there are too many addresses

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_spvid_config

This function maps a Shortest Path VLAN identifier (SPVID) to a base VLAN.

This function implements the `bridge (<1-32> | backbone) spbv bvlan <1-4094> spvid <1-4094>` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_spvid_config(char *bridge_name, u_int16_t mt_id, spb_vid_t bvid, spb_vid_t
spvid)
```

Input Parameters

`bridge_name`	Bridge name
`mt_id`	Multi-topology identifier <3996-4095>
`bvid`	Base VLAN identifier <1-4094>
`spvid`	Shortest Path VLAN identifier <1-4094>

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NAME_IS_NULL` when `bridge_name` is NULL

`RESULT_ERROR` when:

- The bridge instance configuration is NULL
- SPBV is not supported on the bridge
- SPVID automatic allocation is enabled for the bridge instance
- The given `mt_id` does not exist
- The bridge is not configured
- Memory allocation fails
- The backbone VLAN cannot be found

`RESULT_OK` when the function succeeds or when the function fails because SPVID automatic allocation is enabled for the bridge instance configuration

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge configuration exists

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

`SPB_API_ERR_SPBV_NOT_EDGE_BRIDGE` when the bridge is not an edge bridge

`SPB_API_ERR_SPBV_AUTOMODE` when SPVID automatic allocation is enabled for the bridge instance

SPB_API_ERR_SPBV_SPVID_OUTRANGE when spvid is outside the configured SPVID pool range

SPB_API_ERR_SPBV_SPVID_EXIST when spvid is already mapped to bvid

SPB_API_ERR_ARG_NULL when there is an internal error

SPB_API_ERR_BRIDGE_NOT_FOUND when the bridge is NULL

SPB_API_ERR_MAX_VLANS_CONFIGURED when the number of VLANs exceeds the maximum that can be configured

SPB_API_ERR_VLAN_NOT_MAPPED_TO_BRIDGE when a given VLAN is not mapped to a bridge

SPB_API_ERR_MTID_NOT_FOUND when mt_id is NULL or the default MTID is not found

SPB_API_ERR_VLAN_NOT_MAPPED_TO_INST when a given VLAN is not mapped to the bridge instance

SPB_API_ERR_ISID_ARE_MAPPED_TO_BVLAN when an ISID is mapped to a VLAN

SPB_API_ERR_VLAN_NOT_MAPPED_TO_MTID when the VLAN is not mapped to the given mt_id

spb_api_spvid_unconfig

This function unmaps a Shortest Path VLAN identifier (SPVID) from a base VLAN.

This function implements the `no bridge (<1-32> | backbone) spbv bvlan <1-4094>` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_spvid_unconfig(char *bridge_name, u_int16_t mt_id, spb_vid_t bvid)
```

Input Parameters

bridge_name	Bridge name
mt_id	Multi-topology identifier <3996-4095>
bvid	Base VLAN identifier <1-4094>

Output Parameters

None

Return Value

SPB_API_ERR_BRIDGE_NAME_IS_NULL when bridge_name is NULL

SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND when the bridge instance is NULL

RESULT_ERROR when:

- SPBV is not supported on the bridge
- The given mt_id does not exist
- The backbone VLAN cannot be found

SPB_API_ERR_SPBV_NOT_EDGE_BRIDGE when the bridge is not an edge bridge

SPB_API_ERR_SPBV_AUTOMODE when SPVID automatic allocation is enabled for the bridge instance

RESULT_OK when the function succeeds

spb_api_unmap_bvlan_to_bridge_instance

This function unmaps BVLANs (Backbone Virtual Local Area Networks) from an SPB bridge instance.

This function implements the `no bridge (<1-32> | backbone) instance (spbm|spbv) vlan` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_unmap_bvlan_to_bridge_instance (char *bridge_name, spb_vid_t start,
                                        spb_vid_t end, u_int32_t instance_id,
                                        s_int16_t mt_id, s_int8_t ect,
                                        bool_t is_mstp)
```

Input Parameters

`bridge_name`	Bridge name
`start`	Starting VLAN identifier <1-4094>
`end`	Ending VLAN identifier <1-4094>
`instance_id`	Instance identifier; one of these constants from `spbd/spb_types.h`:
`SPB_SPBM_INSTANCE_ID`	Shortest Path Bridging MAC
`SPB_SPBV_INSTANCE_ID`	Shortest Path Bridging VID
`mt_id`	Multi-topology identifier <3996-4095>
`ect`	Equal-cost tree algorithm identifier <1-2>
`is_mstp`	Whether this bridge is part of an MSTP

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `start` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge cannot be found

`SPB_API_ERR_BRIDGE_INSTANCE_NOT_FOUND` when the bridge instance is NULL

`SPB_API_ERR_VLAN_UNMAP_NOT_SUPPORTED` when the given `mt_id` and `ect` are default values and `is_mstp` is false

`SPB_API_ERR_VLAN_NOT_MAPPED_TO_INST` when a given VLAN is not mapped to the bridge instance

`SPB_API_ERR_MTID_NOT_FOUND` when `mt_id` is NULL or the default MTID is not found

`SPB_API_ERR_VLAN_NOT_MAPPED_TO_MTID` when the VLAN is not mapped to the given `mt_id`

`SPB_API_ERR_ISID_ARE_MAPPED_TO_BVLAN` when an ISID is mapped to a VLAN

`SPB_API_ERR_ECT_MAP_IS_WRONG` when the given `ect` is invalid

`RESULT_OK` when the function succeeds

spb_api_unset_gmac

This function removes a group MAC address for a base VLAN.

This function implements the `no bridge (<1-32> | backbone) spbv bvlan <1-4094> group-mac` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_unset_gmac(char *if_name, char *bridge_name, u_char *gmac_addr, spb_vid_t vid )
```

Input Parameters

`if_name`	Interface name
`bridge_name`	Bridge name
`gmac_addr`	Group MAC address in `HHHH.HHHH.HHHH` format
`vid`	Base VLAN identifier <1-4094>

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `if_name` is NULL

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is NULL

`RESULT_ERROR` when the backbone VLAN cannot be found

`RESULT_OK` when the function succeeds

spb_api_unset_hostname

This function deletes the name of the host for the SPB bridge.

This function implements the `no isis-spb hostname` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_hostname (char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is not found

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the instance does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_if_hello_interval

This function sets the interface's hello interval to its default (10 seconds).

This function implements the `no isis-spb hello-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_if_hello_interval (char *if_name, int level)
```

Input Parameters

`if_name`	Interface name
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1
`ISTYPE_L2`	Level 2

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the ISIS interface parameters are not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_if_hello_multiplier

This function sets an interface's hello-multiplier value to its default (3).

This function implements the `no isis-spb hello-multiplier` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_if_hello_multiplier (char *if_name, int level)
```

Input Parameters

`if_name`	Interface name
`level`	Instance level; one of these constants from `spbd/isis-spb/spb_isis_types.h`:
`ISTYPE_L1`	Level 1

`ISTYPE_L2`	Level 2

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_ISIS_SET_ERR_INVALID_IS_TYPE` when the given level is not supported

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the ISIS interface parameters are not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_if_lsp_interval

This function sets an interface's LSP transmission interval to its default of 33 milliseconds.

This function implements the `no isis-spb lsp-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_if_lsp_interval (char *if_name)
```

Input Parameters

`if_name`	Interface name

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the ISIS interface parameters are not configured

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_if_retransmit_interval

This function sets the LSP retransmission interval to its default of 5 seconds.

This function implements the `no isis-spb retransmit-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_if_retransmit_interval (char *if_name)
```

Input Parameters

`if_name`	Interface name

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NAME_NULL` when `if_name` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_IF_PARAM_NOT_FOUND` when the ISIS interface parameters are not configured

`SPB_API_ERR_IF_NOT_FOUND` when the interface does not exist

`SPB_API_ERR_ISIS_IF_NOT_FOUND` when the ISIS interface is not enabled

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_ignore_lsp_errors

This function tells SPB to validate LSP checksums and reject an LSP if it has a checksum error.

This function implements the `no isis-spb ignore-lsp-errors` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_ignore_lsp_errors (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_lsp_gen_interval

This function sets the minimum interval between regenerating the same LSP to its default value of 30 seconds.

This function implements the `no isis-spb lsp-gen-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
```

```
int
spb_api_unset_lsp_gen_interval (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_lsp_max_lifetime

This function sets the maximum LSP lifetime to its default value of 1200 seconds.

This function implements the `no isis-spb max-lsp-lifetime` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_lsp_max_lifetime (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_API_SET_ERR_LT_REFRESH_TIME` when the configured refresh interval is greater than or equal to the default maximum lifetime

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_lsp_refresh_interval

This function sets the LSP refresh interval to its default value of 900 seconds.

This function implements the `no isis-spb lsp-refresh-interval` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_lsp_refresh_interval (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_API_SET_ERR_GT_MAX_LSP_LIFETIME` when the maximum LSP lifetime of the ISIS instance is less than or equal to the default refresh interval

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_overload_bit

This function clears the overload bit of self-LSPs. When the overload-bit is set, the router is not used as a transit or forwarding router during SPF calculation. The router continues to receive LSPs when the overload bit is set.

This function implements the `no isis-spb set-overload-bit` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_overload_bit (char *bridge_name, s_int16_t mt_id)
```

Input Parameters

`bridge_name`	Bridge name
`mt_id`	Multi-topology identifier <3996-4095>

Output Parameters

None

Return Values

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is not found

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_API_ERR_MTID_NOT_FOUND` when the given mt_id is not found

`RESULT_OK` or `SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_spf_interval

This function sets the minimum and maximum exponential backoff delay between the receipt of a topology change and the calculation of the Shortest Path First (SPF) to their default values:

- 500 milliseconds for the minimum exponential backoff delay
- 50,000 milliseconds for the maximum exponential backoff delay

This function implements the `no isis-spb spf-interval-exp` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_spf_interval (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` or `tag` is NULL

`SPB_API_ERR_ISIS_MASTER_NOT_FOUND` when the ISIS master is not found

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_api_unset_spsourceid

This function sets the shortest path source identifier to its default value (0)

This function implements the `no bridge (<1-32> | backbone) spsourceid` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_unset_spsourceid ( char *bridge_name,
                           u_int32_t spsourceid)
```

Input Parameters

`bridge_name`	Bridge name
`spsourceid`	The shortest path source identifier

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NAME_IS_NULL` when `bridge_name` is NULL

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge cannot be found

`RESULT_OK` when the function succeeds

spb_api_unset_spvid_pool

This function sets the Shortest Path VLAN identifier (SPVID) pool range to its default <3600-3999>.

This function implements the `no bridge (<1-32> | backbone) spvid-pool` command.

Syntax

```
#include "spbd/spb_api.h"
int
spb_api_unset_spvid_pool(char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge name

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NOT_FOUND` when `bridge_name` is NULL

`RESULT_ERROR` when:

- The bridge is not configured for SPB or SPBM
- The bridge instance is NULL
- The default starting SPVID is greater than the current starting SPVID and a SPVID less than the default starting SPVID is already being used
- The default ending SPVID is less than the current ending SPVID and a SPVID greater than the default ending SPVID is already being used

`RESULT_OK` when the function succeeds

spb_api_unset_system_id

This function sets the network-wide unique identifier for the ISIS-SPB process to zero.

This function implements the `no isis-spb system-id` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spb_api_unset_system_id (char *bridge_name, char *tag)
```

Input Parameters

`bridge_name`	Bridge name
`tag`	Instance tag

Output Parameters

None

Return Value

`SPB_API_ERR_ARG_NULL` when a parameter is NULL

`SPB_ISIS_API_SET_ERROR` when the ISIS master is NULL

`SPB_API_ERR_ISIS_INSTANCE_NOT_FOUND` when the given instance tag does not exist

`SPB_ISIS_SET_ERR_SYSTEM_ID_NOT_CONFIGURED` when the `system_id` has not been set

`SPB_ISIS_API_SET_SUCCESS` when the function succeeds

spb_cist_disable_port

This function disables SPB on an interface.

This function implements the `bridge (<1-32> | backbone) spb disable` command.

Syntax

```
#include "spbd/spb_port.h"
int
spb_cist_disable_port (struct spb_port *port)
```

Input Parameters

`port`	SPB interface

Output Parameters

None

Return Value

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`RESULT_OK` when the function succeeds or when SPB is already disabled on the interface

spb_cist_enable_port

This function enables SPB on an interface.

This function implements the `bridge (<1-32> | backbone) spb enable` command.

Syntax

```
#include "spbd/spb_port.h"
int
spb_cist_enable_port (struct spb_port *port)
```

Input Parameters

`port`	SPB interface

Output Parameters

None

Return Value

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is not found

`RESULT_ERROR` when the bridge is not enabled

`SPB_API_ERR_IF_NOT_FOUND` when the interface is NULL

`RESULT_OK` when the function succeeds or when SPB is already enabled on the interface

spb_nsm_send_convention_id

This function sets the agreement protocol convention that specifies how to use a computed topology digest to determine whether:

- A neighboring switch is operating with identical network topology information
- Frames may be safely forwarded to the neighbor

This function implements the `bridge (<1-32> | backbone) agreement convention id` command.

Syntax

```
#include "spbd/spb_nsm.h"
s_int32_t
spb_nsm_send_convention_id (char *bridge_name, u_char value)
```

Input Parameters

`bridge_name`	Bridge name
`value`	Agreement protocol convention; one of these constants from the `spb_convention_id_type` enum in `spbd/spb_types.h`:

`SPB_NO_AGREEMENT`

No action will be taken when the topology digest does not match

`SPB_LOOP_FREE_BOTH`

The transmitter of the agreement digest does not forward traffic until the topology digest with the neighbor matches

`SPB_LOOP_FREE_MCAST_ONLY`

The transmitter does not forward multicast traffic and allows all unicast traffic

Output Parameters

None

Return Value

Less than zero when the function fails

Greater than zero when the function succeeds

spb_nsm_send_loop_mitign

This function enables or disables loop mitigation.

This function implements the `bridge (<1-32> | backbone) loop-mitigation` command.

Syntax

```
#include "spbd/spb_nsm.h"
s_int32_t
spb_nsm_send_loop_mitign (char *bridge_name, u_char value)
```

Input Parameters

`bridge_name`	Bridge name
`value`	Whether to enable or disable loop mitigation:
`PAL_TRUE`	Enable loop mitigation
`PAL_FALSE`	Disable loop mitigation

Output Parameters

None

Return Value

Less than zero when the function fails

Greater than zero when the function succeeds

spb_nsm_send_loop_prevention

This function enables or disables loop prevention.

This function implements the `bridge (<1-32> | backbone) loop-prevention` command.

Syntax

```
#include "spbd/spb_nsm.h"
s_int32_t
spb_nsm_send_loop_prevention (char *bridge_name, u_char value)
```

Input Parameters

`bridge_name`	Bridge name
`value`	Whether to enable or disable loop prevention:
`PAL_TRUE`	Enable loop prevention
`PAL_FALSE`	Disable loop prevention

Output Parameters

None

Return Value

Less than zero when the function fails

Greater than zero when the function succeeds

spbi_api_del_mtid

This function deletes a mutli-topology identifier.

This function implements the `no isis-spb multi-topology-id` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spbi_api_del_mtid (char *bridge_name, s_int16_t mt_id)
```

Input Parameters

`bridge_name`	Bridge name
`mt_id`	Multi-topology identifier <3996-4095>

Output Parameters

None.

Return Value

`SPB_API_ERR_BRIDGE_NOT_FOUND` when the bridge is not found

`SPB_API_ERR_MTID_NOT_FOUND` when the given `mt_id` is not found

`SPB_API_ERR_VLAN_MAPPED_TO_MTID` when a VLAN is mapped to the given `mt_id`

`RESULT_ERROR` when the function fails

spbi_api_set_mtid

This function sets a mutli-topology identifier.

The multi-topology feature allows the devices in an SPB area to maintain several parallel logical views of the network topology. The devices exchange topology-specific link state advertisements describing the properties of each link.

This function implements the `isis-spb multi-topology-id` command.

Syntax

```
#include "spbd/isis-spb/spb_isis_api.h"
int
spbi_api_set_mtid (char *bridge_name, s_int16_t mt_id_config)
```

Input Parameters

`bridge_name`	Bridge name
`mt_id_config`	Multi-topology identifier <3996-4095>

Output Parameters

None.

Return Value

`SPB_API_ERR_ARG_NULL` when `bridge_name` is NULL

`SPB_ISIS_API_SET_ERROR` when the bridge instance configuration is NULL

`SPB_API_ERR_CONFIG_UPDATE` when the bridge is NULL but the bridge instance configuration exists

`SPB_API_ERR_MTID_PRESENT` when the given `mt_id_config` is already set

`SPB_API_ERR_MAX_MTID_CONFIGURED` when the maximum number of topologies already exist

`RESULT_ERROR` when memory allocation fails or when the maximum number of topologies already exist

`RESULT_OK` when the function succeeds

Index