ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Edge Virtual 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 Edge Virtual Bridging (EVB).

Audience

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

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 this chapter:

- Chapter 1, *Edge Virtual Bridging API*

Related Documents

The following guides are related to this document:

- *Edge Virtual Bridging Command Reference*
- *Edge Virtual Bridging Configuration Guide*
- *Installation 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 Edge Virtual Bridging API

This chapter describes:

- LLDP (Link-Layer Discovery Protocol) functions for Edge Virtual Bridging (EVB) and Channel Discovery and Configuration Protocol (CDCP)
- Functions for the `ecpd` process which is responsible for handling VDP (VSI Discovery and Configuration Protocol) TLV (type-length-value) elements

Overview

With server virtualization, hypervisors move network infrastructure into the physical server by their use of virtual switches (also called soft switches or vSwitches).

This blurs the line between the domains of the server administrator and of the network administrator:

- Server administrators typically configure the vSwitches but cannot see or change the external network configurations.
- Network administrators cannot configure or debug the vSwitches.

Challenges arising from hypervisors include performance loss and management complexity of integrating software-based vSwitches into existing network management.

An approach to deal with server-network edge challenges and to provide more management insight into networking traffic in a virtual machine is Edge Virtual Bridging (EVB) with Virtual Ethernet Port Aggregator (VEPA) technology.

The EVB approach promotes network management and network service provisioning as close to the edge as possible.

Virtual Ethernet Bridges

A Virtual Ethernet Bridge (VEB) is a virtual Ethernet switch implemented in a virtualized server environment. A VEB mimics a traditional external layer 2 (L2) switch for connecting VMs (virtual machines). VEBs can communicate between VMs on a single physical server, or they can connect VMs to the external network.

As shown in Figure 1-1, the most common implementations of VEBs are software-based vSwitches built into hypervisors.

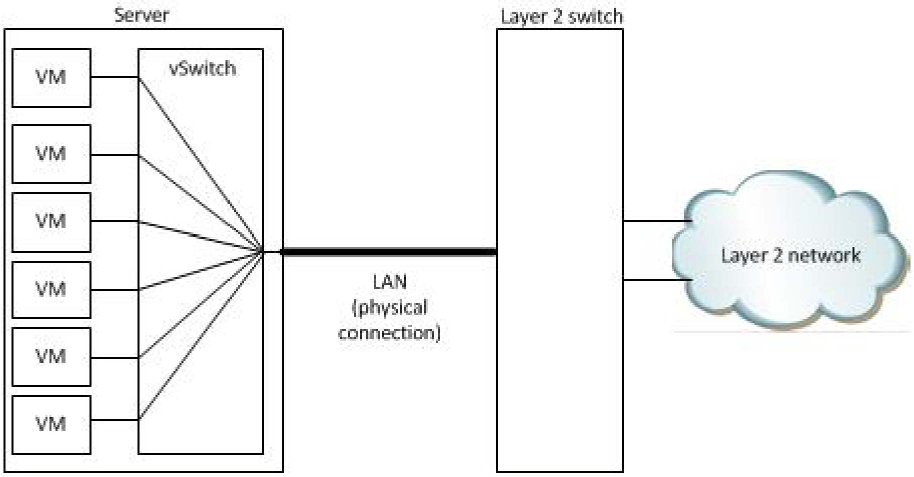

Figure 1-1: vSwitch

In a virtualized server, the hypervisor abstracts and shares physical NICs (network interface cards) among multiple virtual machines, creating virtual NICs for each virtual machine. For the vSwitch, the physical NIC acts as the uplink to the external network. The hypervisor implements one or more software-based virtual switches that connect the virtual NICs to the physical NICs.

Data traffic received by a physical NIC passes to a vSwitch. The vSwitch uses its hypervisor-based configuration information to forward traffic to the correct VMs.

When a VM transmits traffic from its virtual NIC, a vSwitch forwards the traffic in one of two ways:

- If the destination is external to the physical server or to a different vSwitch, the vSwitch forwards traffic to the physical NIC.
- If the destination is internal to the physical server on the same vSwitch, the vSwitch forwards the traffic directly back to another VM.

Using a vSwitch has these advantages:

- Good performance between VMs. A vSwitch can forward internal VM-to-VM traffic directly.
- Deployment without an external switch. Administrators can provide an internal network with no external connectivity.

vSwitches have several disadvantages:

- Consume CPU and memory bandwidth. The higher the traffic load, the greater the number of CPU and memory cycles required to move traffic through the vSwitch, reducing the ability to support larger numbers of VMs in a physical server.
- Lack network-based visibility. vSwitches have a limited feature set and do not provide local traffic visibility or have capabilities for enterprise data monitoring, security, or network management.
- Lack network policy enforcement. Modern external switches have many advanced features such as port security, quality of service (QoS), and access control lists (ACL). But vSwitches often do not have, or have limited support for, such features.
- Lack management scalability. When you increase the number of VMs in a data center, the number of vSwitches also expands. You must manage standard vSwitches individually.

Edge Virtual Bridging

Software vSwitches cannot achieve the level of network capabilities built into enterprise-class L2 data center switches. To solve the management challenges with VEBs, Edge Virtual Bridging (EVB) was developed as the IEEE 802.1Qbg standard. The primary goal of EVB is to combine the best of VEBs with the best of external L2 network switches.

As shown in Figure 1-2, EVB is based on VEPA (Virtual Ethernet Port Aggregator) technology. It is a way for virtual switches to send all traffic and forwarding decisions to an adjacent physical switch. This moves the forwarding decisions and network operations from the host CPU to the switch. It also leverages the advanced management capabilities in the access or aggregation layer switches.

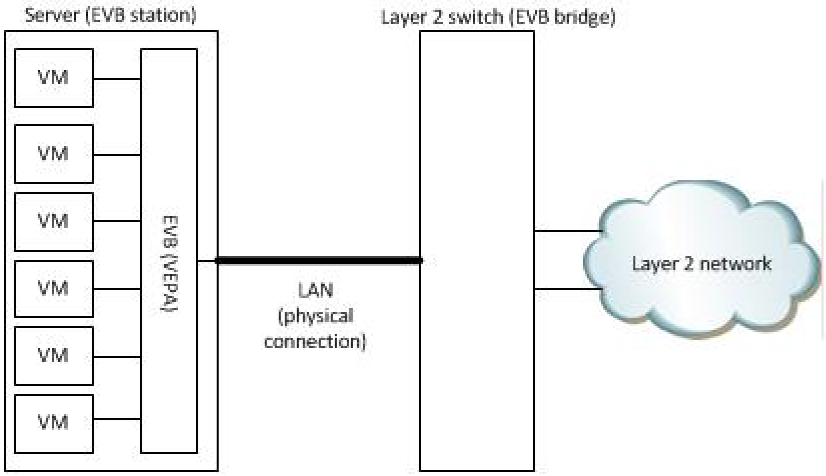

Figure 1-2: EVB using VEPA

Traffic between VMs within a virtualized server travels to the external switch and back through reflective relay (“hairpin” forwarding).

There are many benefits to using EVB/VEPA:

- Reduces the server’s CPU and memory usage from the processing overhead.
- Lets the adjacent switch perform the advanced management functions, so that the NIC can use low-cost circuitry.
- Moves the VM control point into the edge physical switch (top-of-rack or end-of-row switch). VEPA leverages existing investments made in data center edge switching. Administrators can manage the edge network traffic using existing network security policies and tools.
- Gives better visibility and access to external switch features from the guest OS. Network administrators can view frame processing (ACLs) and security features such as Dynamic Host Configuration Protocol guard, address resolution protocol (ARP), ARP monitoring, source port filtering, and dynamic ARP protection and inspection.

Because EVB/VEPA traffic goes deeper into the network, there is some performance reduction. VM-to-VM traffic must flow to the external switch and back—consuming twice the communication bandwidth. This only occurs for co-located VMs on the same host, in the same broadcast domain, and in direct communication with each other. If the need for local bandwidth outweighs the need for visibility or control of network traffic, it makes sense to use VEB mode.

S-channel Technology

S-channel technology adds tagging to VEPA using existing Service VLAN tags (S-Tags) from the Provider Bridging or “Q-in-Q” standard (IEEE 802.1ad). The VLAN tags let you logically separate traffic on a physical interface into multiple channels. Each logical channel (Virtual Station Interface or VSI) operates as an independent connection to the external network.

Figure 1-3 shows how S-channel technology uses these capabilities within a virtualized server.

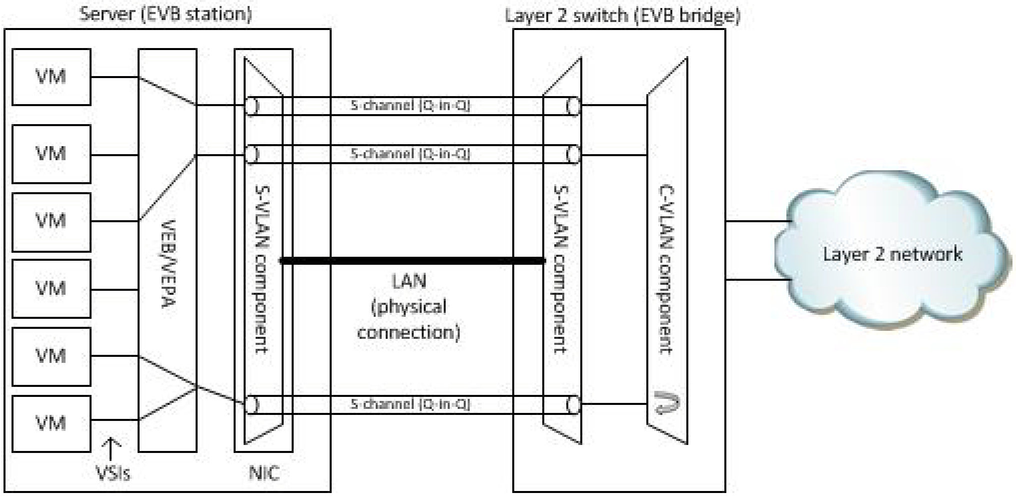

Figure 1-3: S-channel technology

S-channel technology also defines two port-based, link-level protocols shown in Figure 1-4:

- Channel Discovery and Configuration Protocol (CDCP) lets the switch discover and configure the virtual channels. CDCP uses Link-Layer Discovery Protocol (LLDP) and enhances it for servers and external switches.
- Virtual Switch Interface Discovery Protocol (VDP) and Edge Control Protocol (ECP) provide a virtual switch interface that sends attributes for physical and virtual connections to the external switch. VDP/ECP also lets the external switch validate connections and provide the appropriate resources.

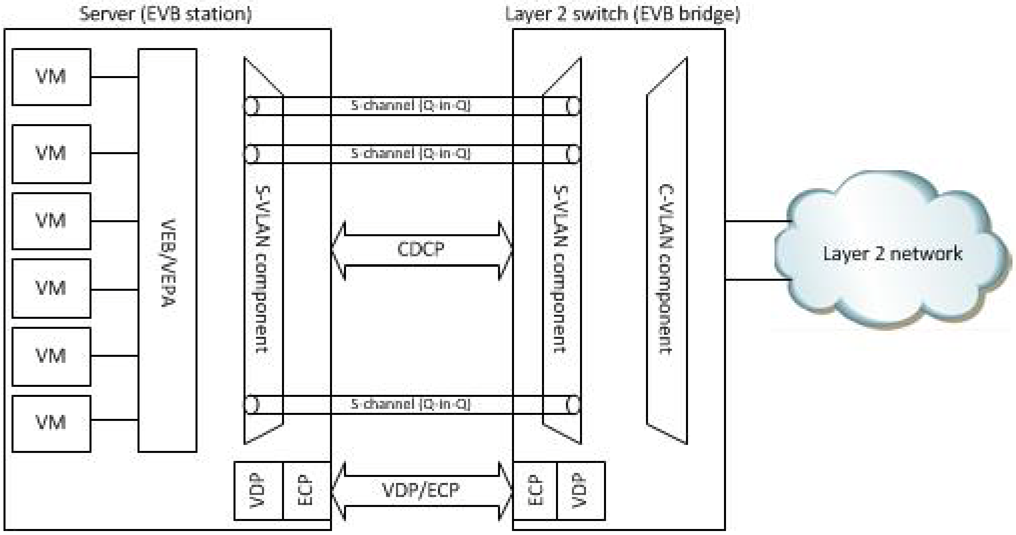

Figure 1-4: CDCP, VDP, and ECP

Data Structures

The functions in this chapter refer to the data structures described in this section.

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

- `cli`
- `interface`

Command API

The functions in this section are called by the commands in the *Edge Virtual Bridging Command Reference*.

Function	Description
cdcp_set_state	Enables or disables CDCP
ecp_api_port_disable	Disables ECP
ecp_api_port_enable	Enables ECP
ecp_api_show_statistics	Displays EVB ECP statistics
ecp_set_acktimer	Sets the ECP acknowledgment time
ecp_set_max_retries	Sets the maximum number of ECP retries
evb_cdcp_set_channel_capacity	Sets the number of CDCP channels
evb_lldp_transmission_mode	Sets the TLV transmission mode
evb_set_reflective_relay	Enables or disables reflective relay
evb_set_svid_pool	Sets the lowest and highest S-VIDs for assignment by CDCP
evb_station_configuration	Configures an EVB station
evb_vdp_set_keep_alive	Sets the VDP keep-alive time
evb_vdp_set_resource_wait_delay	Sets the VDP resource wait delay
vdp_api_show_vsi_info	Displays run time information about the VSIs learned on this station

Include File

You need to include one or more of these header files to call the functions in this section:

- onmd/lldp/lldp_api.h
- ecpd/ecp_api.h

cdcp_set_state

The function enables or disables CDCP (S-Channel Discovery and Configuration Protocol).

This function is called by the `evb cdcp` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
cdcp_set_state (struct interface *ifp, bool_t state)
```

Input Parameters

ifp	Interface
state	Whether to enable or disable CDCP; one of these constants from pal/dummy/pal_types.h:
PAL_TRUE	Enable CDCP
PAL_FALSE	Disable CDCP

Output Parameters

None

Return Values

EVB_ERR_ONM_IF_NOT_EXIST when onmd cannot be found

EVB_ERR_LLDP_IF_NOT_EXIST when lldpd cannot be found

EVB_ERR_PORT_MODE_NO_SUPPORT when the port mode is not SBP or UAP

EVB_ERR_CDCP_ENABLED when CDCP is already enabled on interface

EVB_ERROR when the function cannot allocate memory for cdcp_admin_conf

EVB_ERR_CDCP_DISABLED when CDCP is already disabled on the interface

EVB_SUCCESS when the function succeeds

ecp_api_port_disable

The function disables ECP.

This function is called by the set ecp disable command.

Syntax

```
#include "ecpd/ecp_api.h"
result_t
ecp_api_port_disable(struct interface *ifp)
```

Input Parameters

ifp	Interface

Output Parameters

None

Return Values

ECP_API_ERR_ECP_IF_NOT_EXIST when the interface is NULL or when the ECP interface cannot be found

ECP_API_ERR_ECP_DISABLED when ECP is already disabled on port

ECP_API_ERR_ECPM_NOT_EXIST when the ECP master is NULL

ECP_API_ERROR when the function was unable to disable ECP

ECP_API_SUCCESS when the function succeeds

ecp_api_port_enable

The function enables Edge Control Protocol (ECP).

This function is called by the `set ecp enable` command.

Syntax

```
#include "ecpd/ecp_api.h"
result_t
ecp_api_port_enable(struct interface *ifp)
```

Input Parameters

`ifp`	Interface

Output Parameters

None

Return Values

ECP_API_ERR_ECP_IF_NOT_EXIST when the interface is NULL or when the ECP interface cannot be found

ECP_API_ERR_INTERFACE_NOT_L2 when the ECP interface is not layer 2

ECP_API_ERR_INVALID_BRIDGE when the bridge type is not EVB or SVLAN-EVB

ECP_API_ERR_ECPM_NOT_EXIST when the ECP master is NULL

ECP_API_ERR_ECP_ENABLED when ECP is already enabled on the interface

ECP_API_ERROR when ECP is administratively disabled on the interface

ECP_API_SUCCESS when the function succeeds

ecp_api_show_statistics

The function displays EVB (Edge Virtual Bridging) ECP statistics.

This function is called by the `show evb ecp statistics` command.

Syntax

```
#include "ecpd/ecp_api.h"
result_t
ecp_api_show_statistics(struct interface *ifp, struct cli *cli)
```

Input Parameters

`ifp`	Interface
`cli`	CLI struct

Output Parameters

None

Return Values

ECP_API_SUCCESS when the function succeeds

ecp_set_acktimer

The function sets the Edge Control Protocol (ECP) acknowledgment time.

This function is called by the `evb bridge ecp-acktimer` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
ecp_set_acktimer (char *bridge_name, u_int8_t ecp_timer)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`ecp_timer`	Acknowledgment time <10-20>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

ecp_set_max_retries

The function sets the maximum number of Edge Control Protocol (ECP) retries.

This function is called by the `evb bridge ecp-max-retry` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
ecp_set_max_retries (char *bridge_name, u_int8_t ecp_maxretry)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`ecp_maxretry`	Maximum number of retries <0-7>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

evb_cdcp_set_channel_capacity

The function sets the number of CDCP (S-Channel Discovery and Configuration Protocol) channels.

This function is called by the `evb bridge cdcp channel-capacity` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_cdcp_set_channel_capacity (char *bridge_name, u_int16_t cap_value)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`cap_value`	Number of CDCP channels <1-167>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

DCB_API_SET_ERR_HW_NO_SUPPORT when the function cannot set the capacity

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

evb_lldp_transmission_mode

The function sets the TLV (type-length-value) transmission mode.

This function is called by the `evb bridge tlv-mode` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_lldp_transmission_mode (char *bridge_name, enum evb_tlv_mode mode)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`mode`	Mode; one of these constants from `onmd/onmd.h`:
`EVB_TLV_MODE_MANUAL`	Use the local configuration for EVB
`EVB_TLV_MODE_AUTO`	Determine the configuration by comparing the local and remote LLDP EVB objects

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

evb_set_reflective_relay

The function enables or disables reflective relay.

This function is called by the `evb reflective-relay` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_set_reflective_relay (struct interface *ifp, bool_t reflective_relay)
```

Input Parameters

`ifp`	Interface
`reflective_relay`	
	Whether to enable or disable reflective relay; one of these constants from `pal/dummy/pal_types.h`:
`PAL_TRUE`	Enable reflective relay
`PAL_FALSE`	Disable reflective relay

Output Parameters

None

Return Values

EVB_ERR_ONM_IF_NOT_EXIST when `onmd` cannot be found

EVB_ERR_LLDP_IF_NOT_EXIST when `lldpd` cannot be found

EVB_ERR_PORT_MODE_NO_SUPPORT when the port mode is not SBP or UAP

EVB_ERR_REFLECTIVE_RELAY_ENABLED when reflective relay is already enabled on the interface

EVB_ERR_REFLECTIVE_RELAY_DISABLED when reflective relay is already disabled on the interface

EVB_SUCCESS when the function succeeds

evb_set_svid_pool

The function sets the lowest and highest S-VIDs (Service VLAN identifiers) for assignment by CDCP (S-Channel Discovery and Configuration Protocol).

This function is called by the `evb bridge cdcp svid-pool-range` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
```

```
s_int32_t
evb_set_svid_pool (char *bridge_name, u_int32_t low_value,u_int32_t high_value)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`low_value`	Lowest S-VID <2-4094>
`high_value`	Highest S-VID <2-4094>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the function cannot allocate memory for EVB parameters or when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

evb_station_configuration

The function configures an EVB station.

This function is called by the `evb-station` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_station_configuration (struct interface *ifp,u_int8_t ecp_maxretry,
                            u_int8_t ecp_timer,u_int8_t res_wait_delay,
                            u_int8_t keep_alive,u_int8_t reflective_relay)
```

Input Parameters

`ifp`	Interface
`ecp_maxretry`	Maximum number of ECP retries
`ecp_timer`	ECP acknowledgment time
`res_wait_delay`	VDP (VSI Discovery and Configuration Protocol) resource wait delay
`keep_alive`	VDP keep alive time
`reflective_relay`	
	Whether to enable or disable reflective relay; one of these constants from `pal/dummy/pal_types.h`:
`PAL_TRUE`	Enable reflective relay
`PAL_FALSE`	Disable reflective relay

Output Parameters

None

Return Values

CLI_ERROR when the interface is NULL

EVB_ERROR when `onmd` cannot be found

EVB_ERROR when `lldpd` cannot be found

EVB_ERROR when the function cannot allocate memory for `evb_station`

EVB_SUCCESS when the function succeeds

evb_vdp_set_keep_alive

The function sets the VDP (VSI Discovery and Configuration Protocol) keep-alive time.

This function is called by the `evb bridge vdp-keep-alive` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_vdp_set_keep_alive (char *bridge_name,u_int8_t keep_alive)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`keep_alive`	Keep-alive time <0-31>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

evb_vdp_set_resource_wait_delay

The function sets the VDP (VSI Discovery and Configuration Protocol) resource wait delay.

This function is called by the `evb bridge <1-32> resource-wait-delay` command.

Syntax

```
#include "onmd/lldp/lldp_api.h"
s_int32_t
evb_vdp_set_resource_wait_delay (char *bridge_name, u_int8_t res_wait_delay)
```

Input Parameters

`bridge_name`	Bridge identifier <1-32>
`res_wait_delay`	Resource wait delay <0-31>

Output Parameters

None

Return Values

EVB_ERR_BRIDGE_NOT_FOUND when the bridge is not found

EVB_ERR_BRIDGE_NO_EVB when the bridge is not an EVB bridge

EVB_ERR_EVB_PARAM_NOT_INITIALISED when the EVB parameters have not been set

EVB_SUCCESS when the function succeeds

vdp_api_show_vsi_info

The function displays run time information about the VSIs (Virtual Station Interfaces) learned on this station.

This function is called by the `show evb vdp vsi-info` command.

Syntax

```
#include "ecpd/vdpd.h"
result_t
vdp_api_show_vsi_info(struct cli *cli)
```

Input Parameters

`cli`	CLI structure

Output Parameters

None

Return Values

ECP_API_SUCCESS when the function succeeds

Index