ipinfusion™

ZebOS-XP® Network Platform

Version 1.4

Extended Performance

Virtual Router Redundancy Protocol

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 Virtual Router Redundancy Protocol (VRRP).

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This document contains these chapters and appendices:

- Chapter 1, *Introduction*
- Chapter 2, *Data Structures*
- Chapter 3, *VRRP API*
- Chapter 4, *VRRP SNMP API*

Related Documents

The following guides are related to this document:

- *Virtual Router Redundancy Protocol Command Reference*
- *Virtual Router Redundancy Protocol Configuration Guide*
- *Network Services Module Developer Guide*
- *Network Services Module Command Reference*
- *Installation Guide*

- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 Introduction

This chapter provides an introduction to the Virtual Router Redundancy Protocol (VRRP).

Overview

This section provides an overview of VRRP implementation in ZebOS-XP. Refer to this RFC 5798 (for both IPv4 and IPv6) for details about VRRP.

VRRP enables a virtual router composed of two or more VRRP routers on the same subnet to prevent failure by supplying at least one designated Standby virtual router, if the designated Master virtual router fails. It is designed to eliminate the single point of failure most common in a static default routed environment. The ZebOS-XP VRRP module provides for increased availability of the default gateway servicing hosts on the same subnet.

VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to a VRRP router on a LAN. The VRRP-enabled router controlling the IP addresses associated with a virtual router is called the Master, and it forwards packets sent to these IP addresses. The election process manages dynamic failover in the forwarding responsibility should the Master become unavailable. Any of the virtual router's IP addresses on a LAN can then be used as the default first-hop router by end-hosts. The advantage gained from using VRRP is a higher availability default path without requiring configuration of dynamic routing or router discovery protocols on every end-host.

Typically, end hosts are connected to the enterprise network through a single router (first-hop router) that is in the same Local Area Network (LAN) segment. The most popular method of configuration is for the end hosts to statically configure this router as their default gateway. This minimizes configuration and processing overhead. The main problem with this configuration method is that it produces a single point of failure if this first-hop router fails.

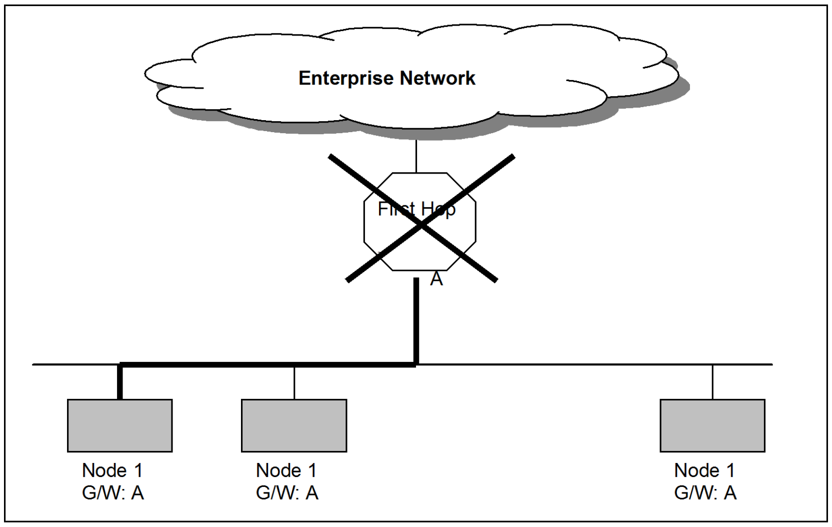

Figure 1-1: Network with No VRRP Configuration

VRRP attempts to solve this problem by introducing the concept of a *virtual* router, composed of two or more VRRP routers on the same subnet. The concept of a *virtual* IP address is also introduced, which is the address that end hosts configure as their default gateway. Only one of the routers (Masters) forwards packets on behalf of this IP address. In the event that the Master fails, one of the other routers (Backups) will assume forwarding responsibility for it.

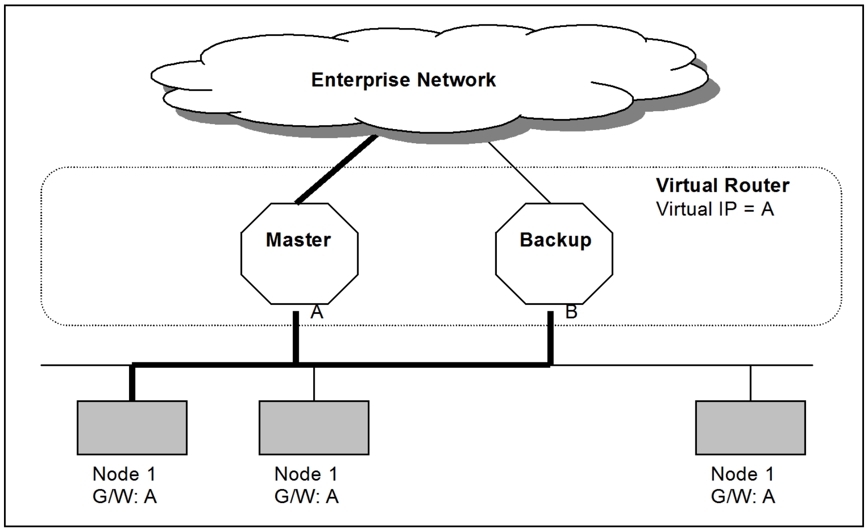

Figure 1-2: VRRP Configuration — Master and Backup Up

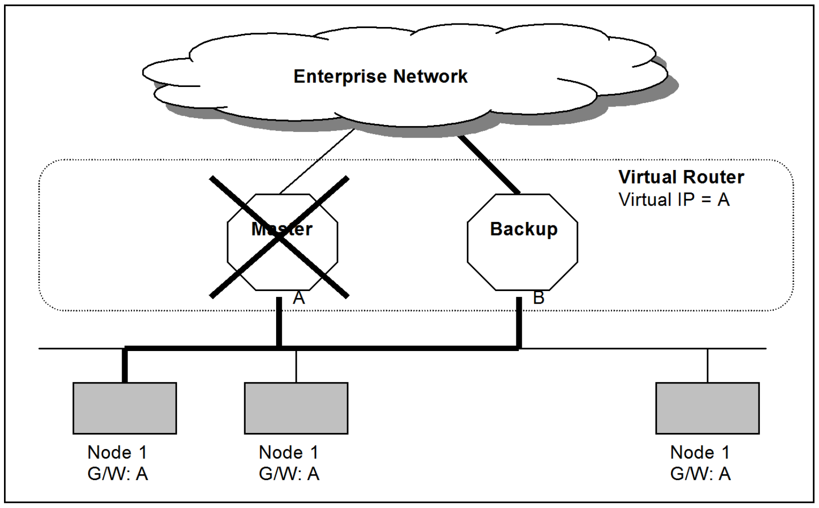

Figure 1-3: VRRP Master-Backup Configuration — Master Down

Using VRRP, you can also configure the two virtual routers (each assigned a virtual IP address) for load sharing so that during normal condition, traffic is evenly split between the routers. Half of the end hosts are configured to use address A as their default gateway, and the other half are configured to use address B. Each of the virtual routers, is the Master for one of the virtual routers and Backup for the other.

The VRRPd acts as IMI clients and registers with IMI. The VRRP maintains its CLI tree for all the VRRP commands.

The VRRPd registers for the ZebOS-XP event notifications for service messages. One of them is event notification from NSM as a service message.Other messages may be internally processed by NSM client (which is a part of VRRPd) and invokes the registered callbacks.

Interface Tracking

The VRRP Interface Tracking feature allows tracking the interface state that can alter the priority level of a virtual router for a VRRP group. For example, an interface can be tracked and if it goes down, then the priority of the VRRP group can be lowered, which may allow another VRRP router to become the new group master virtual router. ZebOS-XP allows tracking of only one interface per VRRP group.

Sub-second Detection

ZebOS-XP supports VRRP Version 3 for IPv4 and IPv6 based on RFC 5798, which allows VRRP to support sub-second detection.

Note: A problem might occur when other routing protocols, such as OSPF, are also enabled on these switches as these protocols are slower to converge than VRRP. This can lead to the situation where the route table of the Master VRRP router is not completely populated when it switches back as the Master. Thus, routes to the remote networks can be lost during the convergence time of the routing protocol resulting in dropped packets for traffic destined to these networks.

Backward Compatibility with VRRPv2

ZebOS-XP supports backward compatibility of VRRPv2 with VRRPv3 using a configuration flag that set or unset with the `v2-compatible` command. When backward compatibility, the virtual router can send and receive both VRRPv2 and VRRPv3 advertisements.

The following applies to the backward compatibility with VRRPv2 feature:

- When a master virtual router supports backward compatibility, it sends both VRRPv2 and VRRPv3 advertisements at the configured advertisement interval. This interval is the length of time, in seconds, between each advertisement sent from the master to its backup(s). If the sub-second detection feature is enabled, the sub-second rate is ignored.
- A backup virtual router that supports backward compatibility ignores VRRPv2 advertisements from the master virtual router, if it is also receives VRRPv3 advertisements from it.
- When a backup virtual router supports backward compatibility and the master virtual router supports only VRRPv2, the backup virtual router translates the received advertisement interval to centiseconds.

Architecture

The VRRP module runs in the context of an NSM process in the ZebOS-XP architecture model.

The VRRP module provides the following main functions of VRRP:

- Accepts VRRP configuration from the CLI, such as VR-ID, timers, VRRP mode, and priority, and configures the sessions accordingly.
- Brings up or brings down the IPv4 or IPV6 sessions in the respective VR mode.
- Allows for the creation and deletion of VRRP sessions dynamically.
- Transmits and receives the VRRP packets to and from the virtual-router peers based on the time-out value or the current state of the VRRP router and performs functions related to the VRRP Finite State Machine (FSM). For more information about FSM, see Section 6, “Protocol State Machine”, of RFC 5798.
- Handles the advertisement messages it receives from the IPv4 module and the IPv6 module.

VRRP functionality is available in the Privileged VR (PVR) context, which is the default VR context.

In addition to the RFC 5798 requirements, the following requirements are met:

- The ZebOS-XP package or operating system (OS) must contain an IP component that supports the sending and receiving of IP Multicast messages. The IP component must pass all messages of type 112 to the VRRP application upon receipt.

- There must be a method of configuration that can be modified or extended to support configuration of the VRRP application.
- The system must provide a one-second timer to signal the VRRP application.
- The forwarding (layer 2) mechanism must:
    - support the sending of gratuitous ARP messages
    - either be able to be extended in order to contain VRRP information, or provide Application Programming Interfaces (APIs) to allow VRRP to modify the ARP table.
- The hardware must be able to accept (pass up) messages whose MAC address does not match any owned physical MAC addresses on the router. If the hardware does not support this requirement, the router can run in promiscuous mode, allowing software to make the decision.

Architectural Strategies

The following strategies take into account considerations of extensibility and integration:

- VRRP manages its own allocation and de-allocation of memory. For VRRP sessions, the VRRP code allocates memory as it is needed. The average number of sessions is five, and the total possible is 255. By allocating memory dynamically, memory is conserved. The overhead of dynamic allocation is compensated by the few number of times that allocation is required.
- VRRP statically creates an array (table) to hold pointers to each of the VRRP sessions. This requires far less memory than statically allocating memory for all sessions, and facilitates access to any VRRP session with complexity O(1).

This section describes the interactions between VRRP and other ZebOS-XP components and lists any of the messages that are exchanged between the VRRP module and each of the other components.

lSystem Architecture

VRRP is an independent protocol module that uses the NSM client/server mechanism to learn the interface, VR, IP address information, and so on from NSM. The protocol module uses the NSM client to register for the relevant services from NSM. The NSM process uses the NSM server module to manage the list of connected of NSM Clients and to publish information to them. As part of the NSM-VRRP split, the NSM client code in VRRPd handles updates from NSM and builds the local interface table. The NSM client/ server messaging mechanism has message types necessary for VRRPd. Common functions are grouped in separate files appropriately. (For example, all communication to and from the IP component are be grouped in the file `vrrp_ip.c`

)nterfaces

VRRP interacts with the following ZebOS-XP components:

- VRRP interacts with NSM, HSL, IMI, Management Interface and SNMP Agent
- Network Services Module (NSM)—corresponds to System Layer
- Forwarding Engine (FE)—corresponds to Forwarding Layer
- Configuration Management—ZebOS-XP CLI or SNMP

VRRP Layer

VRRP is an IP protocol which interacts with IP for sending and receiving advertisement messages (IP multicast). It may also require interaction for initialization.

System Layer

VRRP requires support from the system layer for setting up a timer, and for message passing and queuing facilities.

Forwarding Layer

VRRP interacts with the FE for passing table information and for sending gratuitous Address Resolution Protocol (ARP) messages.

Configuration Management

VRRP requires configuration by the Command Line Interface (CLI), if one exists, or by other configuration application or methods, such as SNMP CLI.

Object Model

This section illustrates the different modules that make up VRRP:

- CLI Module - `vrrp_cli.c`. This file contains the definitions for the CLI commands
- API Module - `vrrp_api.c` This file contains the API functions to facilitate communication between the CM application and VRRP.
- IP Module - `vrrp_ip.c` This file contains API functions to facilitate communication between the IP application and VRRP. In addition, all functions to call any IP API functions are set up here. The called file contains these functions.
- FE Module - `vrrp_fe.c` This file contains API functions to facilitate communication between VRRP and the forwarding layer. In addition, all functions to call any FE APIs are set up here.
- SYS Module - `vrrp_sys.c` This file contains API functions to facilitate communication between VRRP and the operating system.

  Note: Operating system may also refer to a component managing the operating system. This may include functions to set up timers, pass messages between processes, etc.
- VRRP Module - `vrrp.c` This file contains the VRRP backbone functionality, including message processing, initialization, state transition, etc.

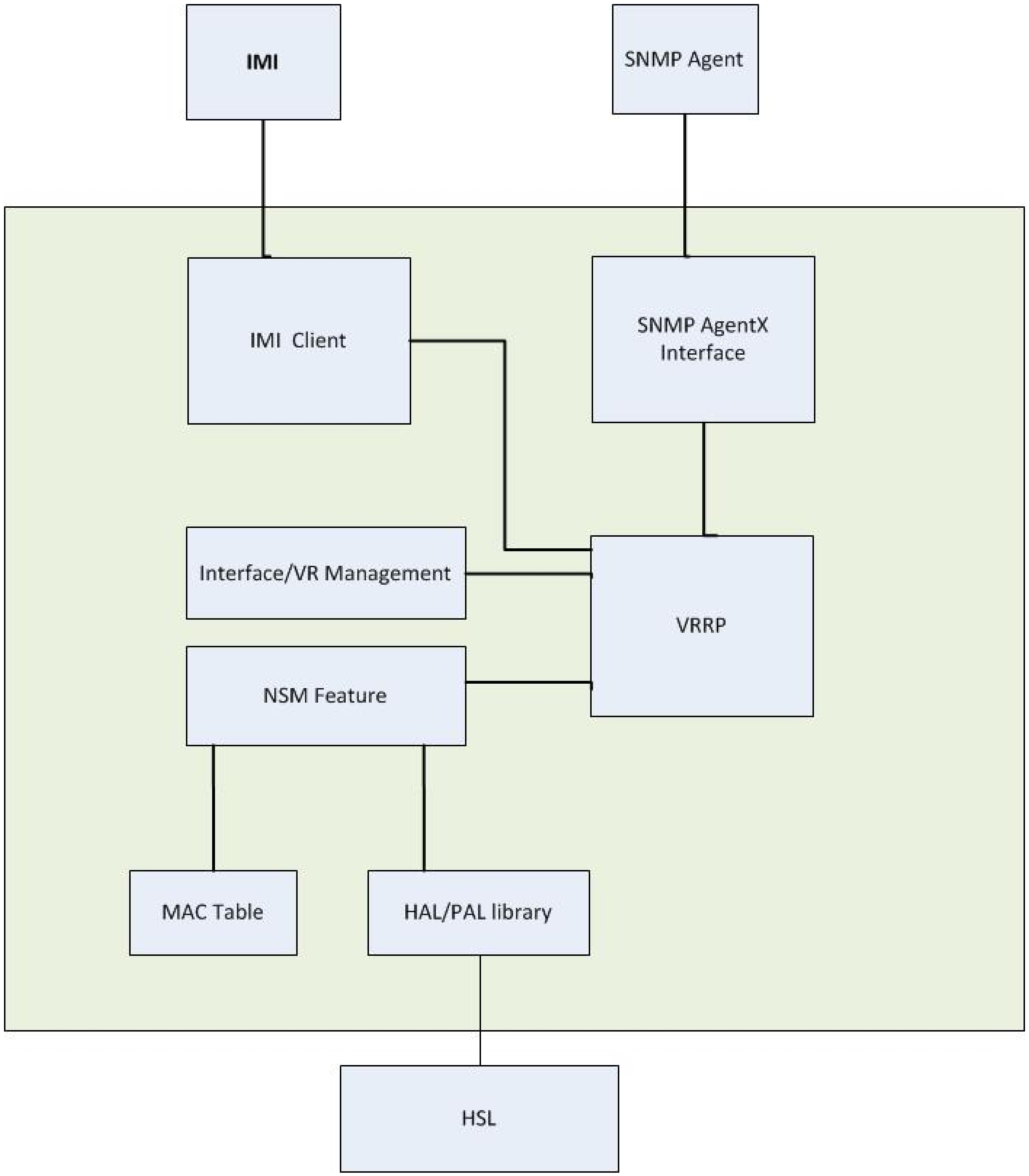

Figure 1-4: Module communication

Modules

The following diagram shows the details of VRRP interaction with system components.

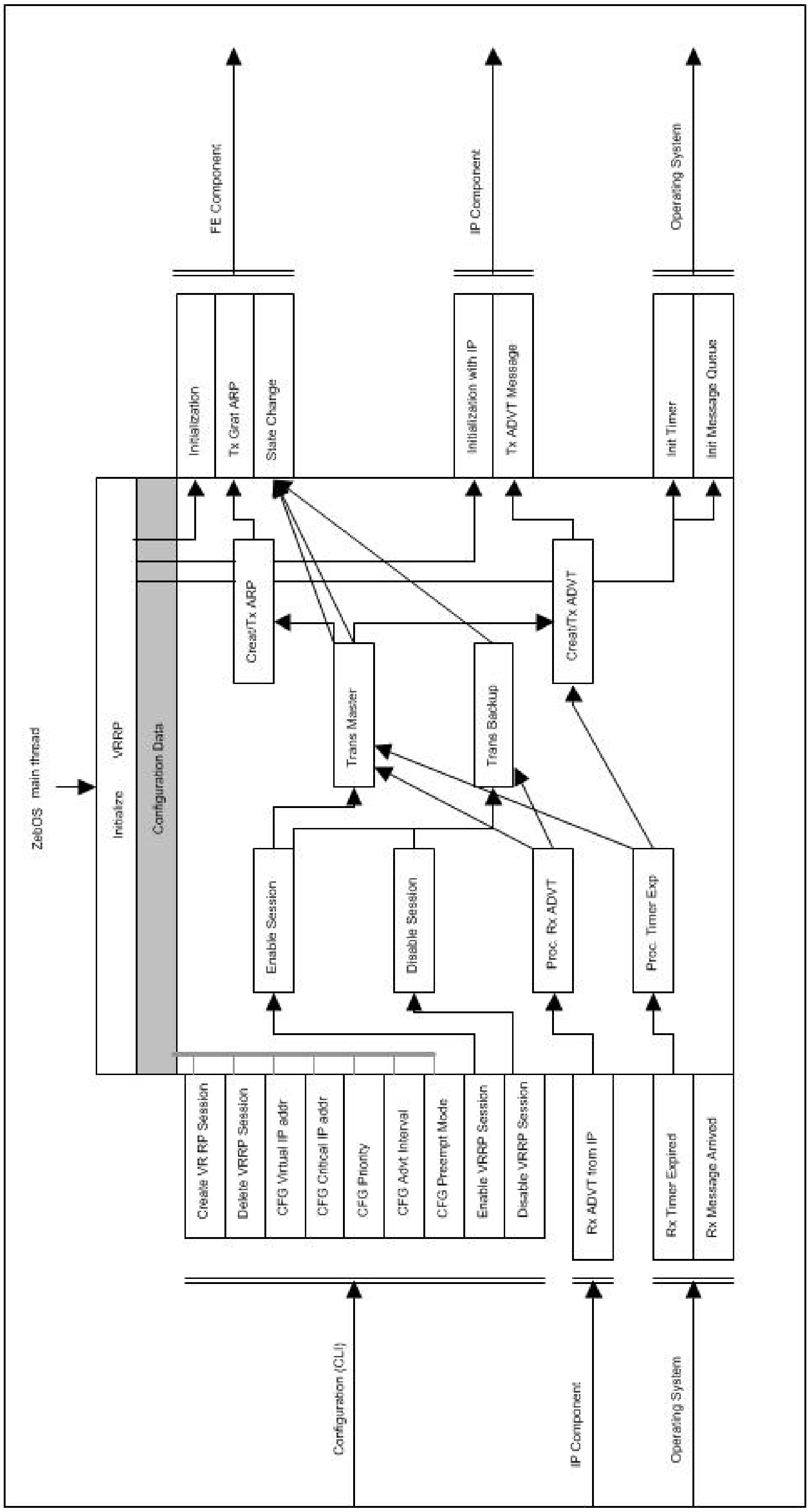

Figure 1-5: VRRP Interaction with System Components

VRRP Module Architecture

Many of the VRRP requirements involve lower-level operating system calls to be implemented correctly. As a result, some of the functionality differs from platform to platform. The majority of the implementation has been made OS-independent, however, this is not possible for all areas.

All operating system-dependent functionality resides in various PAL files, and an API is established between VRRP and PAL. The API definitions for each OS reside in `/pal/<platform>/pal_kernel.c` where <platform> is “linux”, “ose”, “openbsd”, etc.

System Implementation Details

The following files are in the `..\vrrp` directory:

vrrp.c	vrrp.h
vrrp_api.c	vrrp_api.h
vrrp_cli.c	vrrp_fe.c
vrrp_init.c	vrrp_init.h
vrrp_ip.c	vrrp_sys.c
vrrp.tbl	

System Requirements

The following subsections describe the features and functions that must be available before VRRP can operate correctly. It presents each of the three requirements, along with possible implementation strategies, some of which have been employed on Linux.

The Linux implementations of the data plane code are located in `pal/linux/plat_vrrp.c`.

Sending Gratuitous ARP Requests

If a function already exists for sending gratuitous Address Resolution Protocol (ARP) requests, the virtual IP and MAC addresses can be used, as described in the following sections. Also, the appropriate API should be left empty, to avoid having to remove all references to the functions throughout all the code.

ZebOS-XP VRRP creates its own ARP request packets, including Ethernet header, and passes them to the IP stack for transmission.

Under Linux, this functionality is accomplished by creating the appropriate sockets to send "raw" packets.

Forwarding Packets Addressed to the Virtual MAC Address

When a VRRP router is functioning as master for the virtual router, it must forward all packets addressed to the virtual MAC address. Again, there are several options for handling this requirement. They are described below:

Note: Option 1 is used by ZebOS-XP VRRP on Linux.

Option 1: Manipulate the interface MAC address

Setting the interface MAC address to the virtual MAC causes all packets addressed to the virtual MAC to be forwarded correctly: This solves requirement 3. To solve requirement 2, there are two options. First (probably the easiest) is to add the virtual MAC address to a secondary MAC address on to the interface. This causes ARP replies to be generated automatically and correctly. The second option is to use promiscuous mode, and watch for ARP Requests for the virtual IP address. Whenever one is received, an ARP reply can be manually built and sent.

It may not always be possible to manipulate the interface MAC address. If this is the case, or the customer implements a software forwarder, Option 2 should be used.

Option 2: Use promiscuous mode

If the customer implements a software forwarder, this option may be preferred, even if it is possible to manipulate the interface MAC address.

The customer implementation may use promiscuous mode to inspect all packets. Any packet addressed to a virtual MAC address for which the router is currently the master router must be forwarded. All ARP requests for a virtual IP address for which the router is currently master must be replied to with the virtual MAC address.

This method could be implemented by storing a linked-list of virtual IP/virtual MAC pairs, and comparing these against all received packets.

Replying to ARP Requests for the Virtual IP Address

When a VRRP router is functioning as master for the virtual router, it must reply to ARP requests for the virtual IP address with the virtual MAC address. Setting the interface MAC address to the virtual MAC causes all packets addressed to the virtual MAC to be forwarded correctly.

Interacting with NSM Client/Server Module

ZebOS-XP VRRPd interacts with NSM by exchanging TLV-encoded messages. Some services requests needs a response, to match the request with the response message. There are some services which does not require a response, as they are notifications sent by NSM to the VRRPd process to indicate that an important event has occurred.

To interact with NSM, VRRP process needs to create an NSM Client in its process space. Using this NSM Client, it then needs to establish a connection with NSM. When the connection fails, the NSM connection is re-established using the same client. Within the NSM process is an NSM Client-Server that manages client connections and handles messages.

The VRRPd Protocol Daemon registers for multiple ZebOS-XP event notifications. Some of these event notifications come directly from NSM as NSM service messages, while others are internally processed by the NSM Client module which is a part of VRRPd, and then registered callbacks invoked.

VRRP Virtual IP address are installed and uninstalled as the secondary address on the interface in NSM for VRRPv3 when the VRRP is a Master and accept mode is true. NSM implements interface Manager, therefore it needs to find out if the address is a VRRP virtual IP Address or not. The VRRPd send information about its Virtual IP address to NSM once the VRRP is enabled. NSM maintains a list of VRRP Virtual IP address and uses is for address validation during IP address installation/deletion.

NSM force shutdowns all the VRRP session that runs on the interface when it associates it or dissociates it with the Virtual Router. It also force shutdown when the interface is changed from L3 to L2.VRRPd are not needed does not need these calls and are handled independently

CHAPTER 2 Data Structures

This chapter describes the data structures that Virtual Router Redundancy Protocol (VRRP) uses.

Common Data Structures

The following data structures are common for all ZebOS-XP protocols and are used in VRRP functions:

- `interface`
- `lib_globals`

Refer to the *Common Data Structures Developer Guide* for a description of these data structures.

VRRP Data Structure

The data structures described in this section are used in VRRP functions. These structures are found in `vrrp/vrrp.h`.

Chapter 3, *VRRP API*, refers to the data structures described in this section.

vrrp_session

This data structure holds the VRRP session information. It resides in the `vrrp/vrrp.h` file.

Member	Description
`vrrp`	Pointer to VRRP global
`vrid`	Virtual router ID
`state`	Current state of the VRRP session. it is VRRP_STATE_INIT,VRRP_STATE_MASTER, VRRP_STATE_BACKUP
`af_type`	Address family
`vip_v4`	Virtual IP address
`master_ipv4`	Master real IP address
`vip_v6`	Virtual IPv6 address
`master_ipv6`	Master real IP V6 address
`owner`	Indicates whether it is the IP owner
`primary_addr`	The primary ipv4 address of the configured VRRP interface

Member	Description
`adv_src_v4`	Source IPV6 advertisement
`ifindex`	Interface index
`ifp`	Interface pointer points to particular interface
`prio`	Priority
`conf_prio`	Priority of the configuration
`switch_back_delay`	Switchback delay
`switch_back_delay_timer`	Countdown timer for switch_back_delay
`adv_int`	Advertisement interval
`master_adv_int`	Advertisement interval received from Master
`preempt`	Preempt mode
`vmac`	Virtual Mac of PAL
`timer`	Count Down timer
`vip_status`	This will indicate that the VIP is configured or not.
`shutdown_flag`	The flag is true if it is shutdown
`skew_time`	Skew time
`master_down_int`	The time interval during which the advertisment is not received and the master is down.
`num_ip_addrs`	Number of IP addresses
`link_addr_deleted`	VRRP Link address
`monitor_if`	VRRP Circuit Failover
`priority_delta`	Holds information related to VRRP Circuit Failover
`mc_status`	Holds information related to VRRP Circuit Failover
`mc_down_cnt`	Holds information related to VRRP Circuit Failover
`pri_zero_pkts_sent`	Holds information related to VRRP Circuit Failover
`vrrp_uptime`	SNMP variables
`admin_state`	SNMP variables
`rowstatus`	SNMP variables
`vrrp_advt_info ai`	SNMP variables
`primary_ipaddr_flag`	SNMP variables

Member	Description
`storage_type`	SNMP variables
`accept_mode`	SNMP variables
`stats_become_master`	Times state transitioned to master
`stats_advt_rcvd`	Advertisement received
`stats_advt_int_errors`	Advertisement received with wrong interval
`stats_ip_ttl_errors`	Number of packets received with ttl != 255
`stats_pri_zero_pkts_rcvd`	Number of packets received with priority=0
`stats_pri_zero_pkts_sent`	Number of packets sent with priority=0
`stats_invalid_type_pkts_rcvd`	Number of packets received with invalid type
`stats_addr_list_errors`	Number of packets received with address mismatch
`stats_invalid_auth_type`	Number of packets received with unknown authentication type
`stats_auth_type_mismatch`	Number of packets received with authentication type mismatch
`stats_pkt_len_errors`	Total number of packets received with packet length less than length of VRRP header
`stats_disc_time`	Discontinuity time
`mc_up_events`	Monitored circuit up event
`mc_down_events`	Monitored circuit down event
`new_master_reason`	Reason to transit to the master VRRP router
`new_master_reason_next`	Reason to transit that is to set the master VRRP router
`init_msg_code`	Message codes
`garp_pkt_delay`	Gratuitous ARP packet delay
`garp_snd_flag`	Gratuitous ARP flag

Definition

```
/* Structure to hold the VRRP Session information */
struct vrrp_session
struct vrrp_session
{
  /* Pointer to VRRP global. */
  VRRP_GLOBAL_DATA *vrrp;
  /* VRRP Backward compatibility flag */
#ifdef HAVE_VRRP_V3
  bool_t vrrp_version2_compatibility;
#endif  /* HAVE_VRRP_V3 */

  /* VrId. */
  int vrid;

  /* Previous State. */
  vrrp_state_t prev_state;

  /* Current State. */
  vrrp_state_t state;

  /* Address family */
  u_int8_t  af_type;

  /* Virtual IP address. */
  struct pal_in4_addr vip_v4;

  /* Master real IP address. */
  struct pal_in4_addr master_ipv4;

 /* Sender IP address  */
  struct pal_in4_addr sender_ipv4;

#ifdef HAVE_IPV6
  /* Virtual IPv6 address. */
  struct pal_in6_addr vip_v6;
  struct pal_in6_addr master_ipv6;
#endif /*HAVE_IPV6 */
  /* IP owner? */
  bool_t owner;

  /* Configured as Owner */
  bool_t conf_owner;

  /* Primary IPv4 address. */
  struct pal_in4_addr primary_addr;

  /* Source IPv4 address of advertisement. */
  struct pal_in4_addr adv_src_v4;

#ifdef HAVE_IPV6
  /* Source IPv6 address of advertisement. */
  struct pal_in6_addr adv_src_v6;
#endif /* HAVE_IPV6 */

  /* Interface. */
  u_int32_t ifindex;
  struct interface *ifp;

  /* Priority. */
  int prio;
  int conf_prio;

  /*switch back delay */
  int switch_back_delay;
  /* Countdown timer for switch_back_delay */
  int switch_back_delay_timer;

  /* Advertisement interval. */
  int adv_int;
  /*Adv interval received from the master*/
  int master_adv_int;

  /* Preempt mode. */
  bool_t preempt;

  /* Virtual Mac. */
  PAL_MAC_ADDR vmac;
 /* Countdown timer. */
  int timer;

  /* Flags. */
  int vip_status;
  bool_t shutdown_flag;

  /* Skew time. */
  int skew_time;

  /* Master down interval. */
  int master_down_int;

  /* Number of IP addresses. */
  int num_ip_addrs;

#ifdef HAVE_VRRP_LINK_ADDR
  /* VRRP Link address. */
  bool_t link_addr_deleted;
#endif /* HAVE_VRRP_LINK_ADDR */

  /* VRRP Circuit Failover. */
  char  *monitor_if;
  int    priority_delta;
  bool_t mc_status;
  s_int8_t mc_down_cnt;
  bool_t pri_zero_pkts_sent;
  /* SNMP variables. */
  pal_time_t vrrp_uptime;
  int        admin_state;
  int        rowstatus;
  struct     vrrp_advt_info ai;
  int        primary_ipaddr_flag;
  int        storage_type;
  int        accept_mode;

  /* SNMP statistic information */
  int stats_become_master;              /* Times state transitioned to master */
  int stats_advt_rcvd;                  /* Advt rcvd */
 int stats_become_master;              /* Times state transitioned to master */
  int stats_advt_rcvd;                  /* Advt rcvd */
  int stats_advt_int_errors;            /* Advt rcvd with wrong interval */
  int stats_ip_ttl_errors;              /* No. pkts rcvd with ttl != 255 */
  int stats_pri_zero_pkts_rcvd;         /* No. pkts rcvd with priority=0 */
  int stats_pri_zero_pkts_sent;         /* No. pkts sent with priority=0 */
  int stats_invalid_type_pkts_rcvd;     /* No. pkts rcvd with invalid type */
  int stats_addr_list_errors;           /* No. pkts rcvd with addr mismatch */
  int stats_invalid_auth_type;          /* No. pkts rcvd with unknown auth
                                           type */
  int stats_auth_type_mismatch;         /* No. pkts rcvd with auth type mismatch */
  int stats_ip_cnt_mismatch;            /* No. of pkts rcvd with mismatch in
                                           Ip address count */
  int stats_pkt_len_errors;             /* Total num of pkts rcvd with pkt
                                           length < length of VRRP header */
  u_int32_t stats_disc_time;            /* Discontinuity Time */

  int mc_up_events;                      /* Monitored circuit up event. */
  int mc_down_events;                    /* Monitored circuit down event. */

  int new_master_reason;                /* Reason to transit to the master. */
  int new_master_reason_next;           /* Reason to transit that is to set */
  vrrp_init_msg_code_t init_msg_code;
  int garp_pkt_delay;
  bool_t garp_snd_flag;
};
```

vrrp_advt_info

This data structure holds the VRRP packet. It resides in the `vrrp/vrrp.h` file.

Member	Description
ver_type	Version and Type
vrid	Virtual router ID

Member	Description
priority	Priority number assigned to the virtual router
num_ip_addrs	Number of IP addresses to backup
Rsvd:4 and advt_int:12	Reserved and Advertisement interval
auth_type	Authentication type
advt_int	Advertisement interval
cksum	VRRP checksum
vip_v4	Virtual IPv4 address
vip_v6	Virtual IPv6 address
auth_data	Authentication data

Definition

```
* Structure to hold the VRRP Packet. */
struct vrrp_advt_info
{
  /* Version & Type. */
  u_int8_t ver_type;

  /* VrId. */
  u_int8_t vrid;

  /* Priority. */
  u_int8_t priority;

  /* Number of IP addresses to backup. */
  u_int8_t num_ip_addrs;

#ifdef HAVE_VRRP_V3
  /* resvd + Advertisement interval. */
  u_int16_t rsvd:4;
  u_int16_t advt_int:12;
#else
  /* Auth type. */
  u_int8_t auth_type;

  /* Advertisement interval. */
  u_int8_t advt_int;
#endif

  /* VRRP Checksum. */
  u_int16_t cksum;

  /* Virtual IP address. */
  struct pal_in4_addr vip_v4;

#ifdef HAVE_IPV6
  /* Virtual IPv6 address. */
  struct pal_in6_addr vip_v6;
#endif /*HAVE_IPV6 */

#ifndef HAVE_VRRP_V3
  /* Auth data. */
  unsigned int auth_data[2];
#endif
};
```

vrrp_global

This data structure holds data that is common to all the VRRP sessions. It resides in the `nsm/vrrp/vrrp.h` file.

Member	Description
Vm	Back Pointer to VRRP master
notificationcntl	Flag For Trap Generation —1 for Enabled and 2 for Disabled
t_vrrp_timer	VRRP timer
sess_tbl_max	Instance table
sess_tbl_cnt	Instance table
sess_tbl	Instance table
vrrp_sess_cnt	count of VRRP session
asso_tbl_max	Maximum size of the associated table
asso_tbl_cnt	Number of entries of the associated table
asso_tbl	Associated Table
t_vrrp_read	Read thread for ipv4
t_vrrp_ipv6_read	Read thread for ipv6
t_vrrp_prom	Read thread in a promiscous mode for ipv4
t_vrrp_icmpv6_read	Read thread in a promiscous mode for ipv6
sock	Send/Receive socket
ibuf	Stream buffer
i6buf	Stream buffer
icmpv6_buf	Send/receive sockets
l2_sock	VRRP forwarding (layer 2) sockets

Member	Description
sock_net	VRRP forwarding (layer 2) sockets
sock_arp	VRRP forwarding (layer 2) sockets
vrrp_mac	VRRP forwarding (layer 2) sockets VRRP MAC
ipv6_sock	IPv6 socket
proto_err_reason	Indicates reason for error
vrrp_term_debug	Debug flag
vrrp_conf_debug	Debug flag

Definition

```
struct vrrp_global
{
  /* Pointer to NSM master. */
  struct nsm_master *nm;

  /*Flag For Trap Generation */
  int notificationcntl; /* 1 - Enabled, 2 - Disabled */

  /* Instance table. */
#define VRRP_SESS_TBL_INI_SZ 255
  int             sess_tbl_max;
  int             sess_tbl_cnt;
  VRRP_SESSION **sess_tbl;

#ifdef HAVE_SNMP
    /* Instance table. */
#define VRRP_ASSO_TBL_INI_SZ 255
  int             asso_tbl_max;
  int             asso_tbl_cnt;
  VRRP_ASSO    **asso_tbl;
#endif

  /* VRRP timer. */
  struct thread *t_vrrp_timer;

  /* Read thread. */
  struct thread *t_vrrp_read;
  struct thread *t_vrrp_ipv6_read;
  struct thread *t_vrrp_prom;
  struct thread *t_vrrp_icmpv6_read;

  /* Send/Recv socket. */
  int sock;
  struct stream *ibuf;
  struct stream *i6buf;
  struct stream *icmpv6_buf;

  /* VRRP forwarding (layer 2) sockets. */
  int l2_sock;
  int sock_net;
  int sock_promisc;
  int sock_arp;
  PAL_MAC_ADDR vrrp_mac;

  /* VRRP Backward compatibility flag */
#ifdef HAVE_VRRP_V3
  bool_t vrrp_version2_compatibility;
#endif  /* HAVE_VRRP_V3 */

#ifdef HAVE_IPV6
   pal_sock_handle_t ipv6_sock;
   pal_sock_handle_t icmpv6_sock;

#endif /* HAVE_IPV6 */

  int proto_err_reason;

  /* Debug Flags. */
  u_int32_t vrrp_term_debug;
  u_int32_t vrrp_conf_debug;
};
```

CHAPTER 3 VRRP API

This chapter describes the following Virtual Router Redundancy Protocol (VRRP) APIs:

- VRRP Command API on page 29
- VRRP PAL API on page 43

VRRP Command API

The functions described in this section are defined in the `vrrp_api.c` file. These functions are used to facilitate communication between the Configuration Management (CM) application and VRRP.

Function	Description
vrrp_api_accept_mode	Sets the accept mode for a vrrp session when VRPP V3 is enabled
vrrp_api_advt_interval	Sets the advertisement interval for a session
vrrp_api_del_session_by_ifname	Deletes the VRRP session from a specific interface associated with the provided virtual router (VR) ID
vrrp_api_disable_session	Disables a VRRP session
vrrp_api_enable_session	Enables a VRRP session and sets default values that have not been set
vrrp_api_get_session_by_ifname	Retrieves existing VRRP VR session for a specified interface or creates a new one
vrrp_api_lookup_session	Finds the VRRP VR session for the virtual router ID provided.
vrrp_api_monitored_circuit	Sets the monitored circuit for a VRRP session
vrrp_api_preempt_mode	Enables or disables the preempt mode for a session
vrrp_api_priority	Enables the configuration of the priority of the VRRP router for a session
vrrp_api_unset_advt_interval	Sets the advertisement interval to default value
vrrp_api_unset_priority	Sets the priority of the VRRP router to the default value
vrrp_api_virtual_ip	Configures the virtual IP address for a session
vrrp_api_walk_sessions	Walks a VRRP session table and executes a user-provided callback for each non-empty entry
vrrp_api_unset_priority_by_val	Sets the VRRPv2 compatibility flag that enables backward compatibility with VRRPv2 for the virtual router
vrrp_api_switch_back_delay	Sets the switch back delay for the VRRP session
vrrp_api_unset_switch_back_delay	Sets the switch back delay to default
pal_kernel_set_vmac_status	Setting VMAC status to enabled or disabled

Include Files

To use the functions in this section, you must include `/vrrp/vrrp_api.h`.

vrrp_api_accept_mode

This function sets the accept mode for a VRRP session when VRPP V3 is enabled.

Syntax

```
int
vrrp_api_accept_mode (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t ifindex,
                      int mode)
```

Input Parameters

`ipi_vrid`	Virtual router ID (32-bit value)
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index(32-bit value)
`mode`	the value can be 0 or 1. 0 is for accept-mode false 1 is for accept-mode true.

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when the VRRP session does not exist

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_OK when the mode for a VRRP session is set

vrrp_api_advt_interval

This function sets the advertisement interval for a session. It receives the interval and VR ID from the CM application, and configures the appropriate session. This value must be set to the same value on all VRRP routers participating in a session.

Syntax

```
int vrrp_api_advt_interval (int ipi_vrid, u_int8_t af_type, int vrid, u_int32_t
ifindex, int interval);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6).
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index.

`interval`	Advertisement interval in centiseconds for VRRP V3. Input is accepted in multiples of 5 in the range of 5 to 4095. The default value is 100 centiseconds. Advertisement interval in seconds for VRRP v2 is in the range 1 to 255. The default value is 1 second.

Output Parameters

None

Return Values

VRRP_API_SET_ERR_ADVT_INTVL_NOT_FACTOR_OF_FIVE when the advertisement interval is not a multiple of 5 centiseconds

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_OK when the interval for a VRRP session is set

vrrp_api_del_session_by_ifname

This function deletes the VRRP session from a specific interface associated with the provided Virtual Router ID (VR ID). Only disabled sessions can be deleted. This function deallocates the memory for the session.

Syntax

```
int vrrp_api_del_session_by_ifname (int ipi_vrid, u_int8_t af_type, int vrid,
                                    char *ifname);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifname`	Interface name

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_INTERFACE when no VRRP session is found

VRRP_API_SET_ERR_NO_EXIST when the VR ID does not exist

VRRP_API_SET_ERR_ENABLED disable the VRRP session first

VRRP_OK when the VRRP session is deleted from a specific interface

vrrp_api_disable_session

This function disables a VRRP session. It receives the VR ID from the CM application, and calls the disable function `vrrp_shutdown_sess` defined in the VRRP module.

Syntax

```
int vrrp_api_disable_session (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t
ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_API_SET_ERR_DISABLED when the session is not up

VRRP_OK when the session is disabled

vrrp_api_enable_session

This function enables a VRRP session and sets default values that have not been set. It receives the VR ID from the CM application, and calls the enable function `vrrp_enable_sess` defined in the VRRP module.

Syntax

```
int vrrp_api_enable_session (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t
ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when the session is not found

VRRP_API_SET_ERR_ENABLED when the session is active

VRRP_API_SET_ERR_CONFIG_UNSET when the virtual IP address or interface address is not set

VRRP_API_SET_ERR_PRIO_MISMATCH when the priority is not acceptable

VRRP_OK when the session is enabled

vrrp_api_get_session_by_ifname

The function retrieves the existing VRRP VR session for a specified interface or creates a new one.

Syntax

```
int
vrrp_api_get_session_by_ifname (int ipi_vrid, u_int8_t af_type,
int vrid, char *ifname, VRRP_SESSION **pp_sess);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 - 255)
`ifname`	Interface name
`pp_sess`	Pointer to the newly created VRRP session or the retrieved VRRP session if already exists

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_INTERFACE when no such interface is available

VRRP_API_SET_ERR_L2_INTERFACE when a VRRP session cannot be enabled on an L2 interface

VRRP_API_SET_ERR_SEESION_GET_OR_CRE when session creation/retrieval fails

VRRP_OK when the function succeeds

vrrp_api_lookup_session

This function finds the VRRP VR session for the virtual router ID provided. Otherwise, it returns NULL.

Syntax

```
VRRP_SESSION *
vrrp_api_lookup_session (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 - 255)
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_SESSION when the display succeeds

NULL when the display fails

vrrp_api_monitored_circuit

This function sets the monitored circuit for a VRRP session. It handles circuit failover for VRRP session on an interface for a monitored circuit.

Syntax

```
int
vrrp_api_monitored_circuit (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t ifindex,
char *if_str, int priority_delta);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index
`if_str`	Interface name
`priority_delta`	value by which the priority of the VRRP session will get decremented when the monitored circuit goes down

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when the session does not exist

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_API_SET_ERR_NO_SUCH_INTERFACE when the interface does not exists

VRRP_API_ERR_CANNOT_APPLY_INT_TRACK_ON_VRRP_BINDED_INT when VRRP interface tracking cannot be applied on the interface to which the VRRP is bound

VRRP_API_SET_ERR_VIP_UNSET when the virtual IP is not set

VRRP_API_SET_ERR_CANNOT_TRACK_OBJECT when VRRP interface tracking cannot be set for the master virtual router

VRRP_API_SET_ERR_PRIORDELTA_MUST_LSTHAN_CONF_PRIO when the priority delta value is greater than the priority value of the configured VRRP session

VRRP_OK when the monitored circuit for a VRRP session is set

VRRP_API_SET_ERR_PASSIVE_MONITOR_SESSION when the

vrrp_api_preempt_mode

This function enables or disables the preempt mode for a session. It receives the mode and VR ID from the CM application, and configures the appropriate session. This value must be configured the same for all VRRP routers participating in a session. The default value for this variable is PAL_TRUE.

Syntax

```
int vrrp_api_preempt_mode (int ipi_vrid, u_int8_t af_type, int vrid, u_int32_t ifindex,
int mode);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6).
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index.
`mode`	The preempt mode for the VRRP session. A value of 1 results in the preempt mode being set to PAL_TRUE (enabled). Any other value of mode results in the preempt mode being disabled (PAL_FALSE).

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when the VRRP session does not exist

VRRP_API_SET_ERR_ENABLED when the VRRP session is active

VRRP_OK when the preempt mode for a session is set

vrrp_api_priority

This function enables the configuration of the priority of the VRRP router for a session. It receives the priority from the CM application, and configures the appropriate session. If the router is the default Master for the session (for example, it owns the virtual IP address), the priority must be configured as 255. If the router is a backup for the session, the priority must be less than 255. The default priority is 100.

Syntax

```
int vrrp_api_priority (int ipi_vrid, u_int8_t af_type, int vrid, u_int32_t ifindex,
int prio);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6).
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index.
`prio`	Priority for a VRRP session (1 -255). A value of 255 can be assigned only to the session owner.

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when the session does not exist

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_API_SET_ERR_VIP_UNSET when the virtual IP is not set

VRRP_API_SET_ERR_PRIO_CANT_255 when the priority cannot be 255 (a non-owner attempts to set the priority to 255)

VRRP_API_SET_ERR_PRIO_MUST_255 when the priority must be 255 (when the owner attempts to set the priority to a value less than 255)

VRRP_API_SET_ERR_PRIO_MUST_GRTR_DELTA when the priority is less than or equal to the session priority delta

VRRP_OK when the priority for a VRRP session is set

vrrp_api_unset_advt_interval

This function sets the advertisement interval to the default value.

Syntax

```
int vrrp_api_unset_advt_interval (int ipi_vrid, u_int8_t af_type, int vrid,
                                    u_int32_t ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_API_SET_ERR_ENABLED when the session is active

VRRP_OK when the interval for a VRRP session is set

vrrp_api_unset_priority

This function sets the priority of the VRRP router to the default value (VRRP_DEFAULT_IP_OWNER_PRIORITY or VRRP_DEFAULT_NON_IP_OWNER_PRIORITY).

Note: If the router is the default Master for the session (for example, it owns the virtual IP address), the default priority is set as 255. If the router is a backup for the session, the default priority is 100.

Syntax

```
int vrrp_api_unset_priority (int ipi_vrid, u_int8_t af_type, int vrid,u_int32_t
ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_API_SET_ERR_PRIO_CANT_BE_UNSET when a non-owner attempts a priority delta that is greater than or equal to the default non-owner priority

VRRP_OK when the priority for a VRRP session is set

vrrp_api_virtual_ip

This function configures the virtual IP address (both IPv4 and IPv6) for a session. It receives the IP address and VR ID from the CM application, and configures the appropriate session. This function accepts the IP address from the CM application in whatever representation it uses. It assumes that the IP application also accepts addresses using this representation.

Syntax

```
int
vrrp_api_virtual_ip (int ipi_vrid, u_int8_t af_type, int vrid, u_int32_t ifindex,
u_int8_t *vip_addr, bool_t is_owner);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6).
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index.
`vip_addr`	pointer to IPv4 or IPv6 VIP address

`is_owner`	Indicates whether the session is the owner of the IP address (TRUE) or not (FALSE).In other words the configured VIP is its interface address. It is used to determine the session priority to assign (the owner must have a priority of 255, and non-owner may not have a priority of 255).

Output Parameters

None

Return Values

VRRP_OK when the virtual IP is set

VRRP_API_SET_ERR_INVALID_LINKLOCAL_ADDRESS when an IPv6 address that is not a link local address is passed

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session matching the search criteria is found

VRRP_API_SET_ERR_ENABLED when the session is currently active

VRRP_API_VIP_ALREADY_CONFIGURED when the virtual IP address is already configured

VRRP_API_VIP_ALREADY_CONFIGURED_ANOTHER_SESSION when the IP address is already used by another session

vrrp_api_walk_sessions

This function walks a VRRP session table and executes a user-provided callback for each non-empty entry.

Syntax

```
int vrrp_api_walk_sessions (int ipi_vrid, void *ref, vrrp_api_sess_walk_fun_t efun,int
*num_sess);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`vrrp_api_sess_walk_fun_t efun`	
	Callback of vrrp_api_sess_walk_fun_t type

Output Parameters

`ref`	Reference returned with the efun function
`num_sess`	Pointer to the total number of VRRP sessions processed

Return Values

VRRP_OK when the function completes a successful walk of the table

VRRP_FOUND when a search for a single instance succeeds

VRRP_FAILURE when the no object of the search was found

vrrp_api_unset_priority_by_val

This function disables the configuration of the priority of the VRRP router for a session.

Syntax

```
vrrp_api_unset_priority_by_val (int ipi_vrid, u_int8_t  af_type,int vrid,u_int32_t
ifindex,int prio);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Af Type Address Family
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index
`prio`	Priority of the VRRP router for the session

Output Parameters

None

Return Values

VRRP_API_SET_ERR_PRIO_NOT_CONFIGURED when the priority is not configured

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP_OK when the interval for a VRRP session is set

VRRP_API_SET_ERR_PRIO_CANT_BE_UNSET when the priority cannot be unset

vrrp_api_switch_back_delay

This function sets a switch-back delay to default value 0.

Syntax

```
int vrrp_api_switch_back_delay (int ipi_vrid, u_int8_t af_type, int vrid, u_int32_t
ifindex, int switch_back_delay);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`vrid`	VRRP session virtual router ID (1 -255).
`switch_back_delay`	Priority of the VRRP router for the session
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_OK when the interval for a VRRP session is set

vrrp_api_unset_switch_back_delay

This function unsets a switch-back delay.

Syntax

```
int vrrp_api_unset_switch_back_delay (int ipi_vrid, u_int8_t af_type, int vrid,
u_int32_t ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`af_type`	Address Family
`vrid`	VRRP session virtual router ID (1 -255).
`ifindex`	Interface index

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_OK when the interval for a VRRP session is set

vrrp_api_walk_all_sessions

This function refers to all sessions that are active.

Syntax

```
int vrrp_api_walk_all_sessions (int ipi_vrid, void *ref, int af_type,
vrrp_api_sess_walk_fun_t efun, int *num_sess);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`ref`	
`af_type`	Address Family
`vrrp_api_sess_walk_fun_t`	
`vrid`	VRRP session virtual router ID (1 -255).
`num_sess`	Number of sessions

Output Parameters

None

Return Values

VRRP_API_SET_ERR_NO_SUCH_SESSION when no session is found

VRRP_OK when the interval for a VRRP session is set

vrrp_api_set_vmac_status

This function enables or disables Virtual Macs (VMACS).It affects all VRRP groups in the router. On a single network segment, the multiple VRRP groups can be configured, each using a different VMAC. The use of VMAC addressing allows for faster switch over when a backup router assumes the master role.

Syntax

```
int vrrp_api_set_vmac_status( struct vrrp_global *vrrp, int new_vmac_stats);
```

Input Parameters

`vrrp_global`	Virtual router ID.
`vrrp`	
`new_vmac_status`	Enable or disable status of VMAC(Value ‘0’ is disabled and Value ‘1’ is enabled)

Output Parameters

None

Return Values

VRRP_API_MASTER_FOUND when search for session master instance succeeds

VRRP_FAILURE when search for session master instance does not succeeds

VRRP_OK when the interval for a VRRP session is set

vrrp_api_v2_compatibility

This function sets the VRRPv2 compatibility flag that enables backward compatibility with VRRPv2 for the virtual router.

This function is called by the following command:

```
vrrp compatible-v2 (enable|disable)
```

Syntax

```
int vrrp_api_v2_compatibility (int ipi_vrid, u_int8_t  af_type,int vrid, u_int32_t
ifindex,bool_t vrrp_v2_compatibility);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`af_type`	
`vrid`	
`ifindex`	
`vrrp_v2_compatibility`	

Return Values

VRRP_OK when the VRRPv2 compatibility flag is set

vrrp_api_switch_back_delay

This function sets a switch-back delay to default value 0.

Syntax

```
int vrrp_api_switch_back_delay (int ipi_vrid, u_int8_t af_type, int vrid,
                                u_int32_t ifindex, int switch_back_delay);
```

Input Parameters

`ipi_vrid`	Virtual router ID (32-bit value)
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index
`mode`	Accept mode for the VRRP session

Output Parameters

None

Return Values

VRRP_OK when the VRRP session is a success

VRRP_API_SET_ERR_NO_SUCH_SESSION when the session is not active

vrrp_api_unset_switch_back_delay

This function configures the advertisement interval of a virtual router. This is the length of time in seconds, between each advertisement sent from the master to its backup(s).The master virtual router sends VRRP advertisements to other VRRP routers within the same group.

Syntax

```
int vrrp_api_unset_switch_back_delay (int ipi_vrid, u_int8_t af_type, int vrid,
                                      u_int32_t ifindex);
```

Input Parameters

`ipi_vrid`	Virtual router ID (32-bit value)
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)
`vrid`	VRRP session virtual router ID (1 -255)
`ifindex`	Interface index
`mode`	Accept mode for the VRRP session

Output Parameters

None

Return Values

VRRP_OK when the VRRP session is a success

VRRP_API_SET_ERR_ADVT_INTVL_NOT_FACTOR_OF_FIVE when the advertisement interval is not a factor of five.

VRRP_API_SET_ERR_NO_SUCH_SESSION when the session is not active

VRRP_API_SET_ERR_ENABLED when the session is already active

VRRP PAL API

The functions described in this section are only applicable to Linux running on an x86 platform. These functions are defined in `pal/linux/pal_kernel.c`.

API	Application
pal_kernel_gratuitous_arp_send	Sends the provided gratuitous ARP message out through the specified interface
pal_kernel_virtual_ipv4_add	Adds a virtual IPv4 address to the specified interface
pal_kernel_virtual_ipv4_delete	Deletes a virtual IPv4 address from the specified interface
pal_kernel_virtual_ipv6_add	Adds a virtual IPv6 address to the specified interface
pal_kernel_virtual_ipv6_delete	Deletes a virtual IPv6 address from the specified interface
pal_kernel_virtual_mac_add	Adds a virtual MAC address to the specified interface
pal_kernel_virtual_mac_delete	Deletes a virtual MAC address from the specified interface
pal_kernel_vrrp_start	Initializes the platform data for VRRP
pal_kernel_vrrp_stop	Shut down VRRP PAL
pal_kernel_vrrp_advert_send	Send the application provided complete VRRP advertisement packet over specified interface.
pal_kernel_nd_nbadvt_send	Call PAL to send neighbor advertisement
pal_kernel_get_vmac_status	Get Vmac Status
pal_kernel_set_vmac_status	Set the VMAC status

pal_kernel_gratuitous_arp_send

This function sends the provided gratuitous ARP message out through the specified interface.

Syntax

```
result_t pal_kernel_gratuitous_arp_send (struct lib_globals *lib_node, struct stream
*ap, struct interface *ifp,struct vrrp_global *vrrp",);
```

Input Parameters

`lib_node`	Global variables
`ap`	Gratuitous ARP message
`ifp`	The interface pointer

Output Parameters

None

Return Values

RESULT_OK when the message was sent successfully

EVRRP_SOCK_BIND when there was an error binding the socket

EVRRP_SOCK_SEND when there was an error sending the request

pal_kernel_virtual_ipv4_add

This function adds a virtual IP address to the specified interface.

Syntax

```
result_t pal_kernel_virtual_ipv4_add (struct lib_globals *lib_node, struct pal_in4_addr
*vip, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vip`	Virtual IPv4 address
`ifp`	pointer to the interface over which the VIP should be added.
`owner`	Owner status of this address
`vrid`	VRRP virtual router ID

Output Parameters

None

Return Values

RESULT_OK when the insertion of the IP address was successful

VRRP_FAILURE when there was an error with the flags

pal_kernel_virtual_ipv4_delete

This function deletes a virtual IP address from the specified interface.

Syntax

```
result_t pal_kernel_virtual_ipv4_delete (struct lib_globals *lib_node, struct
pal_in4_addr *vip, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vip`	Virtual IP address
`ifp`	The interface
`owner`	Owner status of this address
`vrid`	VRRP virtual router ID

Output Parameters

None

Return Values

RESULT_OK when the deletion succeeds

NSM_API_SET_ERR_MASTER_NOT_EXIST when the NSM master is not found

NSM_API_SET_ERR_ADDRESS_NOT_EXIST when the interface is an IPv4 unnumbered interface

NSM_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

NSM_API_SET_ERR_MUST_DELETE_SECONDARY_FIRST when the function attempts to delete the address from a primary interface before it has been deleted from the secondary interfaces

CLI_ERROR when the interface is not real or not active

NSM_API_SET_ERR_CANT_UNSET_ADDRESS when the address could not be deleted for other reasons not mentioned above

EVRRP_MAC_UNSET when the virtual IP address does not exist

NSM_API_SET_SUCCESS when the function succeeds

NSM_API_SET_ERR_CANT_CHANGE_PRIMARY when the primary address cannot be changed

NSM_API_SET_ERR_CANT_CHANGE_SECONDARY when the secondary address cannot be changed

NSM_API_SET_ERR_CANT_DELETE_DHCLIENT_IP when the DHCP client cannot be deleted

pal_kernel_virtual_ipv6_add

This function adds a virtual IPv6 address to the specified interface.

Syntax

```
result_t pal_kernel_virtual_ipv6_add (struct lib_globals *lib_node, struct pal_in4_addr
*vipv6, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vipv6`	Virtual IPv6 address
`ifp`	pointer to the interface over which the VIP should be added.
`owner`	Owner status of this address
`vrid`	VRRPv6 virtual router ID

Output Parameters

None

Return Values

RESULT_OK when the insertion of the IPv6 address succeeds

VRRP_FAILURE when there was an error with the flags

pal_kernel_virtual_ipv6_delete

This function deletes a virtual IPv6 address from the specified interface.

Syntax

```
result_t pal_kernel_virtual_ipv6_delete (struct lib_globals *lib_node, struct
pal_in4_addr *vipv6, struct interface *ifp, bool_t owner, u_int8_t vrid);
```

Input Parameters

`lib_node`	Global variables
`vipv6`	Virtual IPv6 address
`ifp`	The interface
`owner`	Owner status of this address
`vrid`	VRRPv6 virtual router ID

Output Parameters

None

Return Values

RESULT_OK when the deletion succeeds

EVRRP_MAC_UNSET when the virtual IP address does not exist

pal_kernel_virtual_mac_add

This function adds a virtual MAC address to the specified interface. This MAC address is specified in RFC 5798 to be 00-00-5E-00-01-{VRID} for IPv4 and 00-00-5E-00-02-{VRID} for IPv6.

Note: Under Linux, this function is an empty implementation.

Syntax

```
result_t
pal_kernel_virtual_mac_add (struct lib_globals *zg, u_int8_t vrid, struct interface
*ifp, u_int8_t af_type);
```

Input Parameters

`lib_node`	Global variables
`vrid`	VRRP virtual router ID
`ifp`	The interface
`af_type`	Address family. AF_INET (IPv4) or AF_INET6 (IPv6)

Output Parameters

None

Return Values

RESULT_OK when the MAC could be set

EVRRP_MAC_SET when the MAC could not be set through ioctl

pal_kernel_virtual_mac_delete

This function deletes a virtual MAC address from the specified interface. This MAC address is specified in RFC 5798 to be 00-00-5E-00-01-{VRID} for IPv4 and 00-00-5E-00-02-{VRID} for IPv6.

Note: Under Linux, this function is an empty implementation.

Syntax

```
result_t
pal_kernel_virtual_mac_delete (struct lib_globals *zg, u_int8_t vrid, struct interface
*ifp, u_int8_t af_type);
```

Input Parameters

`lib_node`	Global variables
`vrid`	VRRP virtual router ID
`ifp`	The interface

Output Parameters

None

Return Values

RESULT_OK when the MAC could be set

EVRRP_MAC_SET when the MAC could not be set through ioctl

pal_kernel_vrrp_start

This function initializes the platform data for VRRP.

Syntax

```
result_t pal_kernel_vrrp_start (struct lib_globals *lib_node);
```

Input Parameters

`lib_node`	Global variables

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds

EVRRP_SOCK_OPEN when the socket is open

pal_kernel_vrrp_stop

This function stops the platform data for VRRP.

Syntax

```
pal_kernel_vrrp_stop (struct lib_globals *lib_node);
```

Input Parameters

`lib_node`	Global variables

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds

EVRRP_SOCK_OPEN when the socket is open

pal_kernel_vrrp_advert_send

This function sends the application provided complete VRRP advertisement packet over specified interface.

Syntax

```
pal_kernel_vrrp_advert_send (struct lib_globals *lib_node, struct vrrp_global
*vrrp,struct stream *ap,struct interface *ifp, u_int8_t  af_type)
```

Input Parameters

`lib_node`	Global variables
`ap`	VRRP advert message
`ifp`	Interface pointer

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds else error

pal_kernel_nd_nbadvt_send

This function calla the PAL to send the neighbor advertisement

Syntax

```
pal_kernel_nd_nbadvt_send (struct lib_globals *lib_node,struct stream *ap,struct
interface *ifp,struct vrrp_global *vrrp)
```

Input Parameters

`lib_node`	Global variables
`ap`	VRRP advert message
`ifp`	Interface pointer
`vrrp`	pointer to vrrp global structure

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds else error

EVRRP_SOCK_SEND bind socket to FIB

pal_kernel_get_vmac_status

This function returns the current status of VRRP VMAC flag

Syntax

```
pal_kernel_get_vmac_status (struct vrrp_globals *vrrp);
```

Input Parameters

`vrrp`	Pointer to vrrp global structure

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds else error

pal_kernel_set_vmac_status

This function sets VRRP VMAC flag with the user specified status

Syntax

```
pal_kernel_get_vmac_status (struct vrrp_globals *vrrp, int status);
```

Input Parameters

`vrrp`	pointer to vrrp global structure
`status`	Flag indicating usage of VMAC and RMAC

Output Parameters

None

Return Values

RESULT_OK when the initialization succeeds else error

CHAPTER 4 VRRP SNMP API

This chapter describes the Virtual Router Redundancy Protocol (VRRP) management information base (MIB) implemented in ZebOS-XP.

Overview of MIB Implementation

The MIB support for VRRP is based on draft-ietf-vrrp-unified-mib-06.txt. The VRRP MIB contains the following three groups:

- vrrpOperations Group—Consists of objects related to the configuration and control of VRRP router
- vrrpStatistics Group—Consists of objects containing information useful in monitoring the operation of the VRRP routers
- vrrpNotifications Group—Consists of objects and definitions for use in SNMP notifications the VRRP routers send

Supported Tables

The following tables are included in the VRRP MIB:

- vrrpAssociatedIpAddrTable—Contains the addresses of the virtual router(s) that a given VRRP router backs up.
- vrrpOperationsTable—Contains objects that define the operational characteristics of a VRRP router. Rows in this table correspond to instances of virtual routers.
- vrrpStatisticsTable—Contains the operating statistics for a VRRP router.

vrrpAssociatedIpAddrTable

Objects in this group contains the addresses of the virtual router(s) that a given VRRP router backs up.

Attribute	Syntax	Access	Functions
VRRPASSOCIATEDIPADDR	INETADDRESS	NOACCESS	vrrpAssociatedIpAddrTable
VRRPASSOCIATEDSTORAGETYPE	INTEGER	RCREATE	vrrpAssociatedIpAddrTable
VRRPASSOCIATEDIPADDRROWSTATUS	INTEGER	RCREATE	vrrpAssociatedIpAddrTable

vrrpOperationsTable

Objects in this group define the operational characteristics of a VRRP router. Rows in this table correspond to instances of virtual routers.

Attribute	Syntax	Access	Functions
VRRPOPERATIONSINETADDRTYPE	INTEGER	NOACCESS	vrrpOperationsTable

VRRPOPERATIONSVRID	INTEGER	NOACCESS	vrrpOperationsTable
VRRPOPERATIONSVIRTUALMACADDR	MACADDRESS	RONLY	vrrpOperationsTable
VRRPOPERATIONSSTATE	INTEGER	RONLY	vrrpOperationsTable
VRRPOPERATIONSPRIORITY	GAUGE	RCREATE	vrrpOperationsTable
VRRPOPERATIONSADDRCOUNT	INTEGER	RONLY	vrrpOperationsTable
VRRPOPERATIONSMASTERIPADDR	INETADDRESS	RONLY	vrrpOperationsTable
VRRPOPERATIONSPRIMARYIPADDR	INETADDRESS	RCREATE	vrrpOperationsTable
VRRPOPERATIONSADVINTERVAL	INTEGER	RCREATE	vrrpOperationsTable
VRRPOPERATIONSPREEMPTMODE	INTEGER	RCREATE	vrrpOperationsTable
VRRPOPERATIONSACCEPTMODE	INTEGER	RCREATE	vrrpOperationsTable
VRRPOPERATIONSUPTIME	TIMETICKS	RONLY	vrrpOperationsTable
VRRPOPERATIONSSTORAGETYPE	INTEGER	RCREATE	vrrpOperationsTable
VRRPOPERATIONSROWSTATUS	INTEGER	RCREATE	vrrpOperationsTable

vrrpStatisticsTable

Objects in this group contains the addresses of the virtual router(s) that a given VRRP router backs up.

Attribute	Syntax	Access	Functions
VRRPSTATISTICSMASTERTRANSITIONS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSRCVDADVERTISEMENTS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSADVINTERVALERRORS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSIPTTLERRORS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSRCVDPRIZEROPACKETS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSSENTPRIZEROPACKETS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSRCVDINVALIDTYPEPKTS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSADDRESSLISTERRORS	COUNTER	RONLY	vrrpStatisticsTable
VRRPSTATISTICSPACKETLENGTHERRORS	COUNTERRONLY	RONLY	vrrpStatisticsTable
VRRPSTATISTICSRCVDINVALIDAUTHENTICATIONS	COUNTERRONLY	RONLY	vrrpStatisticsTable
VRRPSTATISTICSDISCONTINUITYTIME	TIMETICKS	RONLY	vrrpStatisticsTable
VRRPSTATISTICSREFRESHRATE	GAUGE	RONLY	vrrpStatisticsTable

MIB Functions

This section describes functions related to reading and writing MIBs.

Function	Description
vrrp_get_oper_master_ipaddr	Retrieves the master IP address of the VRRP virtual router
vrrp_set_asso_ipaddr_rowstatus	Sets the row status of the associated IP address entry
vrrp_set_asso_storage_type	Sets the value of the storage type for associated IP address entry
vrrp_set_notify	Sets the value indicating whether this router generates SNMP notifications
vrrp_set_oper_accept_mode	Sets the accept mode (IPv6 only)
vrrp_set_oper_adv_interval	Sets the time interval between sending advertisement messages
vrrp_set_oper_preempt_mode	Sets the preempt mode to allow the higher priority virtual router to preempt the lower priority master
vrrp_set_oper_primary_ipaddr	Sets the primary IP address of the VRRP virtual router, if multiple associated IP addresses are present
vrrp_set_oper_priority	Sets the operational priority of the VRRP virtual router
vrrp_set_oper_rowstatus	Sets the operational row status of the VRRP virtual router
vrrp_set_asso_storage_type	Sets the value of the storage type for this VRRP virtual router

vrrp_get_oper_master_ipaddr

This function retrieves the master IP address of the VRRP virtual router.

Syntax

```
int vrrp_get_oper_master_ipaddr (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, u_int8_t *ipaddr);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`ipaddr`	Pointer to the location to store the retrieved master IP address

Output Parameters

`ipaddr`	Pointer to the location with the retrieved master IP address

Return Values

VRRP_API_GET_SUCCESS when the function succeeds

VRRP_API_GET_ERROR when the function fails

vrrp_set_asso_ipaddr_rowstatus

This function sets the row status of the associated IP address entry.

Syntax

```
int vrrp_set_asso_ipaddr_rowstatus (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, u_int8_t *ipaddr, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	Row status: 1 - active 2 - notInService 4 - createAndGo 5 - createAndWait 6 - destroy

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_asso_storage_type

This function sets the value of the storage type for associated IP address entry.

Syntax

```
int vrrp_set_asso_storage_type (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, u_int8_t *ipaddr, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`ipaddr`	Associated IP address
`val`	Storage type: 1 - other 2 - volatile 3 - nonVolatile (default) 4 - permanent 5 - readOnly

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_notify

This function sets the value indicating whether this router generates SNMP notifications.

Syntax

```
int vrrp_set_notify (int ipi_vrid, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`val`	Indicates whether the notifications are enabled (VRRP_NOTIFICATIONCNTL_ENA (value of this macro is 1) or disabled (VRRP_NOTIFICATIONCNTL_DIS-value of this macro is 2)

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_accept_mode

This function sets the accept mode (IPv6 only).

Syntax

```
int vrrp_set_oper_accept_mode (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	Accept mode: 1 - true 2 - false (default)

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_adv_interval

This function sets the time interval between sending advertisement messages.

Syntax

```
int vrrp_set_oper_adv_interval(int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	Interval in centiseconds (default: 100)

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_preempt_mode

This function sets the preempt mode to allow the higher priority virtual router to preempt the lower priority master.

Syntax

```
int vrrp_set_oper_preempt_mode (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	Value of the preempt mode: 1 - true 2 - false

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_primary_ipaddr

This function sets the primary IP address of the VRRP virtual router, if multiple associated IP addresses are present.

Syntax

```
int vrrp_set_oper_primary_ipaddr (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, u_int8_t *ipaddr);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`ipaddr`	Pointer to the location storing the selected primary IP address

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_priority

This function sets the operational priority of the VRRP virtual router.

Syntax

```
int vrrp_set_oper_priority (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID.
`aftype`	Address family type (AF_INET or AF_INET6).
`vrid`	VRRP virtual router ID (1 - 255).
`ifindex`	Interface index.
`val`	The value of the operational priority value to be set. For the session owner, the priority must be VRRP_DEFAULT_IP_OWNER_PRIORITY. For a non-owner, the priority must be less than VRRP_DEFAULT_IP_OWNER_PRIORITY.

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_rowstatus

This function sets the operational row status of the VRRP virtual router.

Syntax

```
int vrrp_set_oper_rowstatus (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	The value of the new row status: 1 - active 2 - notInService 4 - createAndGo 5 - createAndWait 6 - destroy

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

vrrp_set_oper_storage_type

This function sets the value of the storage type for this VRRP virtual router.

Syntax

```
int vrrp_set_oper_storage_type (int ipi_vrid, u_int8_t aftype, u_int8_t vrid,
u_int32_t ifindex, int val);
```

Input Parameters

`ipi_vrid`	Virtual router ID
`aftype`	Address family type (AF_INET or AF_INET6)
`vrid`	VRRP virtual router ID (1 - 255)
`ifindex`	Interface index
`val`	Storage type:
	1 - other
	2 - volatile
	3 - nonVolatile (default)
	4 - permanent
	5 - readOnly

Output Parameters

None

Return Values

VRRP_API_SET_SUCCESS when the function succeeds

VRRP_API_SET_ERROR when the function fails

Index