ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Protocol Independent Multicasting
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 Protocol Independent Multicasting (PIM).

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This guide contains these chapters:

- Chapter 1, *Overview*
- Chapter 2, *Process Flow*
- Chapter 3, *Data Structures and Messages*
- Chapter 4, *PIM4 Command API*
- Chapter 5, *PIM6 Command API*

Related Documents

The following guides are related to this document:

- *Protocol Independent Multicasting Command Reference*
- *Multicast Configuration Guide*
- *Installation Guide*
- *Network Services Module Command Reference*

- *Network Services Module Developer 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 Overview

The Protocol Independent Multicasting (PIM) routing protocol helps network nodes that are widely dispersed geographically overcome the distances and protocol differences. The ZebOS-XP PIM module resides in the control plane with other routing protocols.

PIM data is kept in a tree-based database. Join messages add routes to the tree; no route is added without a join message. A key architectural component of a PIM system is a Rendezvous Point (RP). Sending nodes meet receiving nodes through RPs, that is, senders announce their presence to RPs and receivers query RPs for multicast sessions.

PIM is supported in a Virtual Router (VR), with minimal configuration changes, other than the addition of the virtual interface concept to create a mapping from the Virtual Router to the physical resource.

Features

ZebOS-XP PIM features are briefly discussed in the sections that follow.

Unified PIM Process

ZebOS-XP PIM-related processes, including PIM Dense-Mode (DM) and PIM Sparse-Mode (SM), are combined in a single, unified process (`pimd`) that provides the following advantages:

- Maintainability – With only one daemon, there is only one set of tiles to maintain all PIM functionality.
- Better performance – Instead of managing separate PIM processes, one for PIM-SM and one for PIM-DM, the Network Services Module (NSM) needs only to manage one process, reducing the processing requirements of the system.

Source Specific Multicast

ZebOS-XP PIM supports the Source-Specific Multicast (SSM) service model. The network layer service provided by SSM is a channel identified by an SSM destination IP address (G) and a source IP address (S). The source IP transmits datagrams to an SSM destination address (G), and a received can acquire these datagrams by subscribing to the channel (Source, Group, or S, G). PIM utilizes version 2 of the Multicast Listener Discovery (MLD) and version 3 of the Internet Group Management Protocol IIGMP) to support channel subscription.

PIM Sparse Mode

PIM Sparse Mode (PIM-SM: RFC 4601) establishes distribution trees across wide area networks (WAN) by routing packets to multicast groups. PIM-SM constructs a tree from each sender to the receivers in a multicast group and packets from the sender follow the tree to recipients. PIM-SM in multicast groups are thinly populated across a large region. Although it can operate in LAN environments, it is most efficient in WAN environments.

Multicast Source Discovery Protocol

The Multicast Source Discovery Protocol (MSDP) described in RFC 3618 connects multiple IPv4 PIM-SM domains together. MSDP allows multicast sources for a group to be known to all rendezvous points (RPs) in different domains. Each PIM-SM domain uses its own Rendezvous Point (RP) and does not depend on RPs in other domains. An RP runs MSDP over TCP to discover multicast sources in other domains.

An RP in a PIM-SM domain has an MSDP peering relationship with MSDP-enabled devices in another domain. MSDP peers exchange Source-Active (SA) messages to distribute information about sources sending to multicast groups.The receiving RP uses the source lists to establish a source path. The SA cache holds the information for all sources learned through SA messages.

If the multicast sources are of interest to a domain which has receivers, the normal PIM-SM source-tree building mechanism is used to deliver multicast data over an inter-domain distribution tree.

PIM Dense Mode

PIM Dense Mode (PIM-DM: RFC 3973) is a data-driven routing protocol that builds source-based multicast distribution trees that operate on the flood-and-prune principle. PIM-DM uses dense multicast routing and implicitly builds shortest-path trees by flooding multicast traffic domain wide, and then pruning back branches of the tree where no receivers are present. PIM-DM effectively distributes data to target recipients in a concentrated area. PIM-DM is straightforward to implement but generally has poor scaling properties.

PIM Sparse-Dense Mode

PIM Sparse-Dense mode (SM-DM) is a special mode which combines both PIM-SPARSE Mode (PIM-SM:RFC4601) and PIM-Dense mode (PIM-DM: RFC 3793).The inherent properties of SM and DM remains the same. If the domain is configured in SM-DM mode, and if a Rendezvous Point (RP) is configured in the router then for the particular group SM rules are followed.If RP is not configured, DM rules are followed.

Bidirectional PIM

Bidirectional PIM (BIDIR-PIM), is a variant of PIM-SM which dispenses with both encapsulation and source state by allowing packets to be natively forwarded from a source to the RP using shared tree state. These benefits can be observed when using a Many-Many multicast deployment.

Anycast Rendezvous Point

ZebOS-XP PIM implements Anycast Rendezvous Points by:

- Extending the PIM register mechanism
- Using the Multicast Source Discovery Protocol (MSDP)

Embedded Rendezvous Point

ZebOS-XP PIM also supports Embedded RP. This mechanism defines an address allocation policy in which the address of the RP is encoded in a multicast group address. When PIM sparse mode is configured, the embedded RP can be used as the specification for a group-to-RP mapping mechanism.

Group to RP Mapping

ZebOS-XP PIM also supports Group to RP Mapping (RFC 6226). This algorithm is used to choose between several Group-to-RP mappings for a specific multicast group.

Bootstrap Router Mechanism

The Bootstrap Router (BSR) mechanism is used by PIM-capable routers to dynamically obtain information to select rendezvous points (RPs) for multicast groups, instead of requiring administrators to manually key in the RP-group mappings.

PIM ECMP Redirect

ZebOS-XP supports PIM Equal Cost Multipath (ECMP) Redirect (RFC 6754).

The PIM protocol uses the Reverse Path Forwarding (RPF) mechanism to find the upstream interface and router for building the forwarding state. When there are equal cost multiple paths to a RP or Source, the router with the highest address is chosen as the nexthop to build the forwarding state. But, when there are ECMP, using mechanism like hashing or choosing the neighbor with higher IP address do not provide for the spread of traffic among the ECMP. This results in ineffective use of network resources.

PIM ECMP Redirect mechanism helps to improve the RPF procedure over ECMP. It allows path selection to be based on administratively selected metric, such as data transmission delays, path preferences and routing metric.The redirect decision is done only by the upstream routers based on existing flows and per interface flow count.

Note: PIM ECMP Redirect mechanism is supported only in PIM-SM. It is not supported for PIM-DM, BIDIR-PIM and PIM Sparse-Dense mode.

Application Interfaces

PIM code has the following interfaces:

- Command Line Interface (CLI)
- Link (port) Level Support Interface
- IP Stack Interface
- Unicast Routing Table Interface
- Multicast Forwarding Plane Interface
- Internet Group Multicast Protocol (IGMP) Interface

CHAPTER 2 Process Flow

This chapter describes the PIM process flow.

Initialization

(\pimd\pim_main.c, \pimd\pimd.c, \pimd\pimd.h)

`main ()` starts the PIM process from a command line argument string of "`df hp P rv`"; the meaning of each is:

`d, --daemon`	Runs in daemon mode
`f, --config_file`	Set configuration file name
`P, --vty_port`	Set VTY's port number
`v, --version`	Print program version
`h, --help`	Display help and exit

`main()` starts or initializes the following:

- global objects for PIM SM and DM
- PIM feature capabilities check
- signal handler
- access-list
- master thread
- common library functions and global variables
- reads the config file
- initialized PIM-related system and support functions, including debug, CLI, and NSM)
- creates a PIM instance
- starts the `pimd` daemon

NSM Message Handler

(\pimd\pim_vif.h, \pimd\pim_vif.c)

The NSM message handler allocates a virtual interface structure (VIF) and adds virtual interfaces. When NSM sends PIM an event that a hardware interface is up, PIM responds by creating a corresponding Virtual InterFace (VIF), and attaches the VIF to the hardware interface, so that PIM can start running on the interface.

HELLO/Designated Router Mechanism

(\pimd\pim_packet.c, \pimd\pim_neighbor.c, \pimd\pim_vif.c)

These modules select the designated router (DR) mechanism.

Designated Forwarder Mechanism

(\pimd\pim_bidir_df.c, \pimd\pim_bidir_df.h, \pimd\pim_bidir_fsm.h)

These modules select the designated forwarder (DF) mechanism.

Internet Group Management Protocol Mechanism

The IGMPv3 protocol allows a host to join a multicast group, and filter sources from which it wishes to receive or exclude multicast traffic. A host can indicate to receive traffic only from certain sources in a group, or traffic from all except certain sources in a group. The IGMPv3 protocol is provided as a service by NSM. Whenever an IGMPv3 state changes in NSM, it sends an NSM_MSG_IGMP_LMEM message to multicast routing protocols. This message contains the filter mode of either INCLUDE or EXCLUDE and a list of sources for a group. Every message contains the complete IGMPv3 state: hence any additions, deletions, or filter mode changes in the source list must be handled by PIM. This message is handled by a callback which is set as follows:

```
nsm_client_set_callback (nc, NSM_MSG_IGMP_LMEM, pim_nsm_recv_igmp);
```

The callback is declared as follows:

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

PIM maintains local membership information per interface for every group and source to track the IGMPv3 filtering state. The information is in the form of the (*,G) or (S,G) state and the associated filter mode for that state. The (*,G) local information can be in one of two states:

1. PIM_LOCAL_NO_INFO
2. PIM_LOCAL_INCLUDE

Local (*,G) membership in the state, PIM_LOCAL_INCLUDE, results in the creation of the PIM (*,G) TIB state if the router satisfies all conditions required by the pim_include (*,G) macro.

If group G is in the SSM range, the PIM (*,G) TIB state is not created. The (S,G) local membership state can be in one of three states:

1. PIM_LOCAL_NO_INFO
2. PIM_LOCAL_INCLUDE
3. PIM_LOCAL_EXCLUDE

Local (S,G) membership in the PIM_LOCAL_INCLUDE state results in the creation of the PIM (S,G) TIB state if the router satisfies all conditions required by the pim_include (S,G) macro. Local (S,G) membership in the PIM_LOCAL_EXCLUDE state results in the creation of the PIM (S,G,rpt) TIB state if the router satisfies all conditions required by the pim_exclude (S,G) macro. If group G is in the SSM range, the PIM (S,G,rpt) TIB state is not created. All local membership information is deleted when the state transitions to PIM_LOCAL_NO_INFO.

INCLUDE {}

The INCLUDE{} message indicates no hosts are interested in a group, so all (*,G) and (S,G) local membership information states are deleted. The corresponding (*,G), (S,G,rpt) or (S,G) TIB states are also modified or deleted, if required.

INCLUDE {A}

The INCLUDE {A} message indicates hosts are interested in traffic only from sources in list A for a group. This message results in the creation of local (S,G) membership in the PIM_LOCAL_INCLUDE state. Existing local (S,G)

membership in the PIM_LOCAL_EXCLUDE state is converted to the PIM_LOCAL_INCLUDE state. Any existing (*,G) local membership information is transitioned to the PIM_LOCAL_NO_INFO state, and deleted. Any (S,G) local membership information for all sources not in A is also deleted. Processing of this message results in creation of the PIM (S,G) TIB state, and modification or deletion the of PIM (*,G) and (S,G,rpt) TIB states.

EXCLUDE {}

The EXCLUDE {} message indicates hosts are interested in traffic from all sources for group G. Local (*,G) membership is created the with PIM_LOCAL_INCLUDE state, and all local (S,G) membership information is deleted. This message results in the creation of the PIM (*,G) TIB state, and modification or deletion of PIM (S,G) and (S,G,rpt) TIB states.

EXCLUDE {A}

The EXCLUDE {A} message indicates hosts are interested in traffic from all sources for group G, except those listed in list A. Local (*,G) membership is created with the PIM_LOCAL_INCLUDE state. Existing local (S,G) membership information is created or transitioned to the PIM_LOCAL_EXCLUDE state. Any existing (S,G) local membership information of sources not in A is also deleted. Processing of this message results in creation of PIM (*,G) and (S,G,rpt) TIB states, and modification or deletion of PIM (S,G) TIB states.

The IGMPv3 protocol is backward-compatible with IGMPv2 and IGMPv2. IGMPv1 and IGMPv2 information is indicated in the form of EXCLUDE{} and INCLUDE{} messages.

Source-Specific Multicast

PIM allows Source Specific Multicast (SSM) to be configured for either the default range, 232/8, or a non-default range specified via an Access Control List (ACL). When SSM is configured, PIM behavior is modified for all groups in the SSM range as follows:

- No (*,G) or (S,G,rpt) states are created
- No (*,G) or (S,G,rpt) Join/Prune messages are sent
- No Register message is sent from the DR connected to the source
- No packets are forwarded on (*,G) or (S,G,rpt) states
- The RP does not forward any Register-encapsulated packets
- If a Register message arrives at the RP, a Register-Stop message is sent

When SSM is configured, all (*,G) and (S,G,rpt) states for groups in the SSM range are removed.

When SSM configuration is removed, (*,G) and (S,G,rpt) states are created from local IGMP membership information. The (*,G) and (S,G,rpt) states, based on PIM Join/Prune, messages are created when the appropriate message is received.

Join/Prune Mechanism

Join/Prune messages are used by a PIM-capable router to attach or detach itself from a PIM multicast packet distribution tree rooted at the RP or at the DR of the data source, for a particular multicast group. The setting is based on its local host’s interest or downstream participation.

Assert Mechanism

On multi-access networks like Ethernet, two or more PIM routers on a LAN may join different upstream routers due to inconsistent MRIB entries regarding how to reach the Rendezvous Point (RP) or the source. The assert mechanism allows downstream routers, upon noticing a discrepancy, to select a single upstream router as the forwarder.

Internally Generated Events

Ordinarily, events are generated from external sources, for example, Timer expirations, data received, and others and they are fed to a Finite State Machine (FSM). The following is a list of all internally-generated events, that are result of a macros changing their values and the results must be fed to corresponding FSMs at the appropriate time.

- (*,*,RP) Upstream FSM JoinDesired (*,*,RP) going TRUE
- (*,*,RP) Upstream FSM JoinDesired (*,*,RP) going FALSE
- (*,G) Upstream FSM JoinDesired (*,G) going TRUE
- (*,G) Upstream FSM JoinDesired (*,G) going FALSE
- (S,G) Upstream FSM JoinDesired (S,G) going TRUE
- (S,G) Upstream FSM JoinDesired (S,G) going FALSE
- (S,G,rpt) Upstream FSM PruneDesired (S,G,rpt) going TRUE
- (S,G,rpt) Upstream FSM PruneDesired (S,G,rpt) going FALSE
- (S,G,rpt) Upstream FSM RPTJoinDesired (S,G,rpt) going FALSE
- (S,G,rpt) Upstream FSM RPF (S,G,rpt) becoming equal to RPF'(*,G)
- (S,G,rpt) Upstream FSM inherited_olist (S,G,rpt) going non-NULL
- (S,G) Assert FSM CouldAssert (S,G,I) going FALSE
- (S,G) Assert FSM AssertTrackingDesired (S,G,I) going FALSE
- (*,G) Assert FSM CouldAssert (*,G,I) going FALSE
- (*,G) Assert FSM AssertTrackingDesired (*,G,I) going FALSE
- (S,G) Register FSM CouldRegister (S,G,) going TRUE
- (S,G) Register FSM CouldRegister (S,G,) going FALSE

Join/Prune States

The following sections describe the creation and deletion of Join/Prune states.

Creating (S,G) State

When any of the following occurs or becomes true, and there is no existing (S,G) state, it is created:

- joins(S,G) received (+) local_receiver_includes(S,G) goes non-NULL
- Assert(S,G) received
- There are DR(S) that receive (S,G) datagram the Keepalive Timer (KAT) is started
- An RP receives first Register Tunneled packet and wants to Join(S,G)

Deleting (S,G) State

At designated routers (DR) the Register State goes away:

- JoinDesired (S,G) becomes FALSE
- Assert (S,G) state becomes NO INFO NULL
- KAT expires

Creating (*,G) State

(*,G) state is created when any of the following becomes true and, if there is no (*,G) state, it is created:

- Join(*,G) received
- local membership(*,G) goes non-NULLr
- Assert(*,G) is received and there is (*,*,RP) state

Deleting (*,G) State

(*,G) state is deleted when:

- JoinDesired (*, G) becomes FALSE and Upstream Join/Prune State is in NotJoined
- RP changes

Creating (S,G,rpt) State

(S,G,rpt) state is created when it does not exist and any of the following occurs or becomes true:

- A Prune(S,G,rpt) is received
- (S,G) state is created
- For each existing (S,G), when (*, *, RP) or (*, G) is created

Deleting (S, G, rpt) State

(S, G, rpt) state is deleted when it exists and all of the following become true:

- (S, G) state does not exist
- PruneDesired (S, G, rpt) becomes FALSE, or RPT JoinDesired (G) becomes FALSE
- All downstream (S, G, rpt) states are freed

Creating (*,*,RP) State

(*,*,RP) state is created when the following occurs and the state does not exist:

- ZebOS-XP receives a Join(*, *, RP) for an existing RP

Deleting (*,*,RP) State

(*,*,RP) state is deleted when any of the following occurs:

- JoinDesired (*, *, RP) becomes FALSE

- The RP is not in the new RP Set

Register Mechanism

The register mechanism provides for support of multiple Rendezvous Points at the designated router:

1. When a packet arrives from a directly-connected source and the forwarder does not have a forwarding entry, the PIM daemon gets a NOCACHE message.
2. The PIM daemon initializes a Register state and also a (S,G) state. The register interface is added to the outgoing list of the (S,G) forwarding entry in the kernel.
3. The kernel then forwards the packet on the (S,G) interface. When the register interface is in the OLIST of the (S,G) entry, all the kernel does is sends a WHOLEPKT message up to the PIM daemon. The WHOLEPKT message contains the whole IP multicast data packet.
4. The PIM daemon receives the WHOLEPKT message, looks up the RP for the group and then sends a unicast Register message to the RP.

For MSDP, the PIM daemon communicates with the BGP daemon via IPC to get AS number and path information for the peer RPF checks.

CHAPTER 3 Data Structures and Messages

This chapter contains the data structures for the PIM protocol and its processes. It also contains the IPv4 and IPv6 messages used by PIM and MRIBD to communicate with each other.

PIM Data Structures

Common Data Structures

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

- `cli`
- `pal_in4_addr`
- `pal_in6_addr`

PIM Globals Structure

The PIM Globals structure maintains process-wide global variables for the PIM daemon process.

Definition

```
struct pim_globals
{
  u_int32_t capabilities;
#define PIM_CAPABILITY_IPV6                    (1 << 0)
#define PIM_CAPABILITY_PIMSM                   (1 << 1)
#define PIM_CAPABILITY_PIMDM                   (1 << 2)
#define PIM_CAPABILITY_ALL                     ((1 << 3) - 1)

#ifdef HAVE_PIM_MSDP_API
  struct pim4_msdp_globals *msdp;
#endif /* HAVE_PIM_MSDP_API */

#ifdef HAVE_PIM_MBR
  struct pim_mbr *mbr;
#endif /* HAVE_PIM_MBR */
};
```

PIM Master Structure

The PIM master structure carries per-VR level instance information.

Definition

```
struct pim_master
```

```
{
  /* Pointer to VR.  */
  struct ipi_vr *vr;

  /* Pointer to globals.  */
  struct lib_globals *zg;

  /* PIM VR Create Time.  */
  u_int32_t start_time;

};
```

PIM VRF Structure

PIM supports multiple instances, one per VRF instance. The PIM VRF structure carries information per each VRF instance, triggered by the NSM VR IPC message.

Definition

```
struct pim_vrf
{
  /* Pointer to IPI VRF.  */
  struct ipi_vrf *iv;

  /* PIM VIF Address Family (AF_INET/AF_INET6) */
  u_int32_t family;

  /* PIM flags.  */
  u_int32_t flags;
#define PIM_VRF_FLAG_UP                   (1 << 0)
#define PIM_VRF_FLAG_NO_MSG2MRIB          (1 << 1)
#ifdef HAVE_PIM_IPV6
#define PIM_FLAG_EMBED_RP                 (1 << 2)
#endif /* HAVE_PIM_IPV6 */

  /* PIM configuration.  */
  u_int32_t configs;
#define PIM_CONFIG_JP_TIMER               (1 << 0)
#define PIM_CONFIG_IGNORE_RP_SET_PRIORITY (1 << 1)
#define PIM_CONFIG_SPT_SWITCH             (1 << 2)
#define PIM_CONFIG_REG_SRC_ADDR           (1 << 3)
#define PIM_CONFIG_REG_SRC_INTF           (1 << 4)
#define PIM_CONFIG_REG_RATE_LIMIT         (1 << 5)
#define PIM_CONFIG_REG_RP_REACH           (1 << 6)
#define PIM_CONFIG_RP_REG_KAT             (1 << 7)
#define PIM_CONFIG_REG_SUPP               (1 << 8)
#define PIM_CONFIG_RP_REG_FILTER          (1 << 9)
#define PIM_CONFIG_CISCO_REG_CKSUM        (1 << 10)
#define PIM_CONFIG_SSM_DEFAULT            (1 << 11)
#define PIM_CONFIG_SSM_ACL                (1 << 12)
#define PIM_CONFIG_DISCARD_ENTRY2MRIB     (1 << 13)
```

```
#define PIM_CONFIG_EMBED_RP_CONF          (1 << 14)
#define PIM_CONFIG_EMBED_RP_UNCONF        (1 << 15)
#define PIM_CONFIG_MULTICAST_ROUTING      (1 << 16)
#define PIM_CONFIG_BIDIR_ENABLE           (1 << 17)
#define PIM_CONFIG_DF_OFFER_INTERVAL      (1 << 18)
#define PIM_CONFIG_DF_OFFER_LIMIT         (1 << 19)
#define PIM_CONFIG_DF_OFFER_INTERVAL_MSEC (1 << 20)
#define PIM_CONFIG_ROUTER_ID              (1 << 21)

  /* PIM instance start time, for snmp trap */
  pal_time_t start_time;

  /* PIM Tree Information Base.  */
  struct ptree *tib;

  /* Array of all of VIFs.  */
  int vifnum;
  int vifnum_sm;
  int vifnum_dm;
#ifdef HAVE_PIM_SMDM
   int vifnum_smdm;
#endif/*HAVE_PIM_SMDM*/

  vector vifs;
#define PIM_INIT_VIF_VECTOR_SIZE   (32)

  /* PIM nexthop cache.  */
  struct route_table *nexthop_cache;

  /* PIM nexthop timer.  */
  struct thread *t_nexthop_scan;

  /* Input/output buffer.  */
  struct stream *ibuf;
  struct stream *obuf;

  /* Control message buffer for the pim[46]_vrf_recv_packet(). */
  char *cmsg;
  size_t cmsglen;

  /* Default addresses.  */
  struct pim_vrf_addrs *addrs;

  /* Address family specific operations.  */
  struct pim_vrf_methods *m;

  /* PIM socket.  */
  int sock;

  /* Read threads. */
```

```
  struct thread *t_read;

  /* Counters.  */
  u_int32_t mrt_counter[PIM_MRT_TYPE_MAX];
  u_int32_t mrt_vif_counter[PIM_MRT_TYPE_MAX];
#ifdef HAVE_SNMP
  int pim_out_asserts;
  int pim_in_asserts;
  struct pim_last_assert *last_assert;
  struct pim_trap_objects *trap_object;
#endif /* HAVE_SNMP */

  /* -- Remainder of pim_vrf is SM-specific */
  /* Configured JP periodic timer value.  */
  u_int32_t jp_timer;

  /* Timer to send the JP bulk message. */
  struct thread *t_jp_message_flush;

  /* Register VIF. */
  struct pim_vif *register_vif;

  /* Bitmap: bit is set if I am DR on the vif */
  mcastbitmap_t  i_am_dr_list;

  /* PIM Register State tree.  */
  struct ptree *reg;

  /* PIM static RP tree. */
  struct route_table *rp_static;

  /* Anycast-rp structure */
  struct ptree *anycast_rps;

  /* Count of "override" configured static RPs */
  int ovrd_rp_cnt;

#ifdef HAVE_PIM_BIDIR
  /* Count of bidir configured static RPs */
  int bidir_rp_cnt;
  struct route_table *bidir_rp_info;
#endif /* HAVE_PIM_BIDIR */

  struct pal_in4_addr router_id;  /* PIM Router ID used in hello */
  struct pal_in4_addr router_id_config; /* Configured Router ID */
#ifdef HAVE_PIM_ECMP_REDIRECT
  struct list *ecmp_bundle_list;
#endif /* HAVE_PIM_ECMP_REDIRECT */

  /* List of static RP configuration */
```



```
  struct pim_rp_static_conf *st_rp_head;
  struct pim_rp_static_conf *st_rp_tail;

  /* PIM BSR information.  */
  struct pim_bsr *bsr;

#ifdef HAVE_PIM_IPV6
  /* Embedded group to RP mapping AVL tree */
  struct ptree *embed_rp;
#endif /* HAVE_PIM_IPV6 */

  /* Global register timer values shared by RPs and DRs */
#define DFT_REGISTER_SUPPRESSION_TIME 60
 /* Default Register probe time is 5 seconds. Setting this to 20 to avoid
    unnecessary RST timeout when Reg-Stop is not received within 5 seconds,
    in high load conditions */
#define DFT_REGISTER_PROBE_TIME 5
#define DEF_KEEPALIVE_PERIOD 210
  u_int16_t reg_suppression;
  u_int16_t keepalive;
  u_int16_t reg_probe;
  /* The KAT for (S,G) state at the RP initiated by Register message */
  u_int16_t rp_reg_kat;

  /* Access list for SPT switchover */
  char *spt_switch;
  /* Register source address */
  struct prefix reg_src;

  /* Register source interface name */
  char *reg_ifname;

  /* Rate limit for Register messages */
  u_int16_t reg_rate_limit;

  /* Access list for filter Registers at RP */
  char *rp_reg_filter;

  /* Access list for sending Cisco Register checksum */
  char *cisco_reg_filter;

  /* Access list for SSM group range definition */
  char *ssm_grp_rng;

  /* SSM Group Range table. */
  struct route_table *ssm_grp_tbl;
  struct route_table *ssm_grp_tbl;

  u_int32_t term_debug_pim;
  u_int32_t conf_debug_pim;
```

```
  /* List of pointers to (S,G,rpt) nodes which are in PRUNE TEMP or
     PRUNE PENDING TMP state. These are used to generate the event "End
     of message for G" when a multicast group block in a Join/Prune
     message has been parsed completely.  */
  struct list *pim_tmp_sgrpt;

#ifdef HAVE_PIM_MSDP_API
   struct msdp *msdp;
#endif /* HAVE_PIM_MSDP_API */

#ifdef HAVE_PIM_SMDM
   struct ptree *ip_dm_members;
#ifdef HAVE_IPV6
   struct ptree *ipv6_dm_members;
#endif/*HAVE_IPV6*/
#endif/*HAVE_PIM_SMDM*/

  struct pim_stat stat;
  u_int32_t jp_bulk_num_groups;
  u_int32_t jp_bulk_time;
#ifdef HAVE_PIM_BIDIR
  u_int32_t df_offer_interval;
  u_int32_t df_offer_limit;
  u_int32_t df_timer_ophigh;
#endif /* HAVE_PIM_BIDIR */
};
```

PIM VIF Structure

For each PIM interface, the router determines if it is the Designated Router on that interface. It sends PIM Hello messages on each interface and processes Hello messages received from its neighbors on that interface. As a result, the router keeps a list of neighbors on that interface. It also remembers which neighbor is the DR for that interface. Finally, it keeps the configured Join/Prune interval, the interface index, and IGMP related information.

The interface table is an array indexed by interface index. There is a special, single Register VIF in the system, with flag PIM_VIF_FLAG_REGISTER, that is used for register tunnel purposes.

Definition

```
{
  /* Belongging PIM VRF.  */
  struct pim_vrf *vrf;

  /* Physical interface.  */
  struct interface *_ifp;

  /* PIM VIF mode.  */
  pim_vif_mode_t _mode;

  /* VIF index.  When this index is negative, VIF is not yet
     registered.  */
```



```
  int _index;

  /* Configure.  */
  u_int32_t _config;
#define PIM_VIF_CONFIG_MODE              (1 << 0) /* PIM mode.  */
#define PIM_VIF_CONFIG_PRIORITY          (1 << 1) /* DR priority.  */
#define PIM_VIF_CONFIG_HELLO_INTERVAL    (1 << 2) /* Hello interval.  */
#define PIM_VIF_CONFIG_HELLO_HOLDTIME    (1 << 3) /* Hello holdtime.  */
#define PIM_VIF_CONFIG_EXCLUDE_GENID     (1 << 4) /* Gen-id exclude.  */
#define PIM_VIF_CONFIG_BSR_BORDER        (1 << 5) /* PIM domain border.  */
#define PIM_VIF_CONFIG_UNICAST_BSM       (1 << 6) /* Process unicast BSM.  */
#define PIM_VIF_CONFIG_PROPAGATION_DELAY (1 << 7) /* Propagation Delay.  */
#define PIM_VIF_CONFIG_NBR_FILTER        (1 << 8) /* Neighbor filter.  */
#define PIM_VIF_CONFIG_SRO_INTERVAL      (1 << 9) /* State refresh interval. */
#define PIM_VIF_CONFIG_BIDIR_NBR_FILTER  (1 << 10) /* BIDIR Neighbor filter.  */
#define PIM_VIF_CONFIG_ECMP_BUNDLE       (1 << 11) /* ECMP bundle */

  /* Flags.  */
  u_int32_t _flags;
#define PIM_VIF_FLAG_ACTIVE              (1 << 0) /* Active interface.  */
#define PIM_VIF_FLAG_PASSIVE             (1 << 1) /* Passive interface.  */
#define PIM_VIF_FLAG_SOCK_JOINED         (1 << 2) /* Joined to Mcast group.  */
#define PIM_VIF_FLAG_ADDR                (1 << 3) /* Primary address.  */
#define PIM_VIF_FLAG_ADDR_GLOBAL         (1 << 4) /* Global address.  */
#define PIM_VIF_FLAG_REGISTER            (1 << 5) /* Register vif.  */
#define PIM_VIF_FLAG_DR                  (1 << 6) /* Designated router.  */
#define PIM_VIF_FLAG_TRIG_HELLO          (1 << 7) /* Triggered Hello.  */
#define PIM_VIF_FLAG_PERIODIC_HELLO      (1 << 8) /* Periodic Hello.  */
#define PIM_VIF_FLAG_FIRST_HELLO_PERIOD  (1 << 9) /* Fist hello period.  */
#define PIM_VIF_FLAG_SR_TLV              (1 << 10) /* SR-TLV support.  */
#define PIM_VIF_FLAG_LPD_TLV             (1 << 11) /* LPD-TLV support.  */
#define PIM_VIF_FLAG_NO_MSG2MRIB         (1 << 12) /* No MSG to MRIB needed. */
#define PIM_VIF_FLAG_SR_CAPABLE          (1 << 13) /* SR capability.  */
#define PIM_VIF_FLAG_LPD_ENABLED         (1 << 14) /* LPD.  */
#define PIM_VIF_FLAG_NBR_GEN_ID_CHG      (1 << 15) /* Gen ID change.  */
#define PIM_VIF_FLAG_WAIT                (1 << 16) /* Waiting for MRIB update */
#define PIM_VIF_FLAG_NO_DF_ELECTION      (1 << 17) /* DF election can take place or not
*/
/* Avoid thread events during (dm->sm) mode change*/
#define PIM_VIF_MODE_CHANGE              (1 << 18)

  /* Mode specific operations.  */
  struct pim_vif_methods *m;

  /* VIF Primary Address.  */
  struct prefix addr;
  struct prefix addr_global;

  /* Hello interval.  */
  u_int32_t hello_interval;
```

```
  /* Hello holdtime.  */
  u_int32_t hello_holdtime;

  /* Hello variables which will be used for the packet transmission.  */
  struct pim_packet_hello hello;

  /* Default Propogation Delay interval
     draft-05 specifies 0.5 sec, but since our minimum timer resolution
     is 1 sec this value is being set to 1 sec.  */
#define PIM_T_LAN_DELAY_DEFAULT 1

  /* PIM neighbors.  */
  struct route_table *neighbors;

  /* Neighbor filter access list */
  char *nbr_flt;

#ifdef HAVE_PIM_BIDIR
  /* BIDIR neighbor filter access list: controls df election */
  char *bidir_nbr_flt;
#endif /* HAVE_PIM_BIDIR */

#ifdef HAVE_PIM_ECMP_REDIRECT
  /* ECMP bundle name */
  char *ecmp_bdl_name;
  /* For ECMP traffic load balancing. The VIF with the
     least flow count will be chosen for new joins */
  u_int32_t ecmp_flow_count;
#endif /* HAVE_PIM_ECMP_REDIRECT */

  /* VIF DR address.  */
  struct prefix dr_addr;

  /* VIF Local Membership Table, Key: (G) */
  struct ptree *local_members;

  /* SR Message generation interval (sec) for PIM-DM.  */
  u_int16_t sro_interval;

  /* VIF SR Message TTL Min. Threshold (# of hops) */
  u_int8_t vif_sr_ttl_min;

  /* SM Effective Propogation Delay */
  u_int16_t t_effective_prop_dly;

  /* DM VIF Propagation-delay value (millisec) */
  u_int16_t vif_pd;

  /* SM Effective Override interval */
  u_int16_t t_effective_ovrd_int;
```



```
  /* DM VIF Override-Interval timer value (millisec) */
  u_int16_t vif_oi;

  /* DM VIF PruneLimit-Interval timer value(millisec)*/
  u_int16_t vif_pi;

  /* DM VIF GraftRetry-Interval timer value(millisec)*/
  u_int16_t vif_gi;

  /* Hello timer.  */
  struct thread *t_hello;

  /* Triggered Hello timer.  */
   int trig_hello_delay;
#define PIM_TRIGGERED_HELLO_DELAY 5
  struct thread *t_triggered_hello;

  /* per interface timer intervals.  */
  u_int32_t t_periodic; /* used for jt_timer */
#define PIM_T_PERIODIC_DEFAULT 60
#define PIM_T_PERIODIC_INITIAL 7
  u_int16_t jp_holdtime; /* used in jp header */

  /* Randomized delay to prevent implosion of override messages */
  u_int16_t t_override;

  /* Supression perios */
  u_int16_t t_suppressed;

  /* Default Override Interval value draft-05 specifies 2.5 sec, but
     since our minimum timer resolution is 1 sec this value is being
     set to 3 sec.  */
#define PIM_T_OVERRIDE_DEFAULT  3

  /* Per interface Assert time values */
  u_int16_t t_assert_override_int;
#define PIM_T_ASSERT_OVERRIDE_DEFAULT   3
  u_int16_t t_assert_time;
#define PIM_T_ASSERT_TIME_DEFAULT       180
};
```

PIM VIF Method Table

PIM VIF method table hides the PIM mode as well as the address-family-specific method/function call, for example, sending out a PIM hello packet, from the rest of the PIM process. This makes the PIM core code both PIM mode and address-family independent as possible.

Definition

```
struct pim_vif_methods
```

```
{
  int (*sock_join) (const struct pim_vif *vif);
  int (*sock_leave) (const struct pim_vif *vif);
  int (*add_index) (struct pim_vif *vif);
  int (*delete_index) (struct pim_vif *vif);
  bool_t (*is_directly_reachable_addr) (struct pim_vif *vif,
                                        const struct prefix *addr,
                                        const bool_t primary_only);
  int (*add_address) (struct pim_vif *vif, struct connected *ifc);
  int (*delete_address) (struct pim_vif *vif, struct connected *ifc);
  bool_t (*update_address) (struct pim_vif *vif);
  struct prefix * (*get_address) (struct pim_vif *vif);
  struct prefix * (*get_address_global) (struct pim_vif *vif);
  int (*start) (struct pim_vif *vif);
  int (*stop) (struct pim_vif *vif);
  int (*register_stop) (struct pim_vif *vif);
  int (*add_success) (struct pim_vif *vif, u_int16_t vif_id);
  int (*delete_success) (struct pim_vif *vif);
  int (*make_passive) (struct pim_vif *vif);
  int (*hello_timer) (struct pim_vif *vif);
  int (*dr_selection) (struct pim_vif *vif);
  struct pim_vif *(*lookup_received_vif) (struct pim_vif *vif,
                                          u_int32_t ifindex, u_int16_t flags);
  int (*send_packet) (struct pim_vif *vif, const struct prefix *dst,
                      const u_char *data, const u_int32_t length);
  int (*send_packet_hello) (struct pim_vif *vif);
  int (*send_packet_register) (struct pim_vif *vif, const struct prefix *dst,
                               const u_int32_t reserved2,
                               const struct pim_sg_prefix *sg);
  int (*send_packet_register_stop) (struct pim_vif *vif,
                                    const struct prefix *src,
                                    const struct prefix *dst,
                                    const struct pim_sg_prefix *sg);
  int (*send_packet_join) (struct pim_vif *vif, const struct prefix *ups_addr,
                           struct pim_mrt *mrt);
  int (*send_packet_prune) (struct pim_vif *vif, const struct prefix *ups_addr,
                            struct pim_mrt *mrt);
  int (*send_packet_assert) (struct pim_vif *vif, const struct prefix *src,
                             const struct prefix *grp, const bool_t xg_assert,
                             const u_int32_t pref, const u_int32_t metric);

#ifdef HAVE_PIM_ECMP_REDIRECT
  int (*send_packet_ecmp_redirect) (struct pim_vif *vif,
                                    const struct pim_sg_prefix *sg,
                                    struct prefix *nbr_addr,
                                    struct pim_interface_id *iid,
                                    u_int8_t preference, u_int64_t *metric);

  int (*process_packet_ecmp_redirect) (struct pim_vif *vif, struct prefix *src,
                                       struct prefix *dst);
```



```
#endif /* HAVE_PIM_ECMP_REDIRECT */

  int (*process_packet) (struct pim_vif *vif, struct prefix *src,
                         struct prefix *dst);
  int (*process_packet_hello) (struct pim_vif *vif, struct prefix *src,
                               struct prefix *dst);
  int (*process_packet_register) (struct pim_vif *vif, struct prefix *src,
                                  struct prefix *dst);
  int (*process_packet_register_stop) (struct pim_vif *vif, struct prefix *src,
                                       struct prefix *dst);
  int (*process_packet_jp) (struct pim_vif *vif, struct prefix *src,
                            struct prefix *dst);
  int (*process_packet_bsm) (struct pim_vif *vif, struct prefix *src,
                             struct prefix *dst);
  int (*process_packet_assert) (struct pim_vif *vif, struct prefix *src,
                                struct prefix *dst);
  int (*process_packet_graft) (struct pim_vif *vif, struct prefix *src,
                               struct prefix *dst);
  int (*process_packet_graft_ack) (struct pim_vif *vif, struct prefix *src,
                                   struct prefix *dst);
  int (*process_packet_crp) (struct pim_vif *vif, struct prefix *src,
                             struct prefix *dst);
  int (*process_packet_state_refresh) (struct pim_vif *vif, struct prefix *src,
                                       struct prefix *dst);
  int (*send_add) (struct pim_vif *vif);
  int (*send_delete) (struct pim_vif *vif);
  int (*send_mrt_state_refresh_flag_update) (struct pim_vif *vif,
                                             struct pim_sg_prefix *sg);
  int (*process_member) (struct pim_vif *vif, struct pim_vif_member *m);
  int (*process_nocache) (struct pim_vif *vif, struct pim_sg_prefix *sg);
  int (*process_wrongvif) (struct pim_vif *vif, struct pim_sg_prefix *sg);
  int (*process_wholepkt_req) (struct pim_vif *vif, struct pim_sg_prefix *sg,
                               u_int32_t message_id);
  /* MBR alert methods.  */
  int (*process_delete_alert) (struct pim_vif *vif);
  int (*process_nbr_add_alert) (struct pim_vif *vif);
  int (*process_nbr_delete_alert) (struct pim_vif *vif);

#ifdef HAVE_PIM_BIDIR

  int (*process_packet_df_elec) (struct pim_vif *vif, struct prefix *src,
                                 struct prefix *dst, u_int8_t subtype);

  int (*send_packet_df) (struct pim_vif *vif, const struct prefix *src);
#endif /* HAVE_PIM_BIDIR */
};
```

PIM Neighbor Structure

PIM neighbor is the PIM mode and address-family aware PIM neighbor instance created through PIM hello control packet processing.

Definition

```
sstruct pim_neighbor
{
  /* Neighbor's route node.  */
  struct route_node *rn;

  /* Flags.  */
  u_int32_t flags;
#define PIM_NBR_FLAG_DR                         (1 << 0)
#define PIM_NBR_FLAG_DELETED                    (1 << 1)
#define PIM_NBR_FLAG_HELLO_HOLDTIME_RCVD        (1 << 2)
#define PIM_NBR_FLAG_HELLO_LPD_RCVD             (1 << 3)
#define PIM_NBR_FLAG_HELLO_GEN_ID_CHG           (1 << 4)
#define PIM_NBR_FLAG_HELLO_SR_CAP_RCVD          (1 << 5)
#define PIM_NBR_FLAG_BIDIR                      (1 << 6)

  /* PIM VIF.  */
  struct pim_vif *vif;

  /* Neighbor's address.  */
  struct prefix addr;

  /* Neighbors' Uptime.  */
  pal_time_t uptime;
  pal_time_t exptime;

  /* Neighbors' Generation ID.  */
  u_int32_t gen_id;

  /* Neighbor hello message parameter.  */
  struct pim_packet_hello hello;

  /* Neighbors' SR Interval (secs) */
  u_int8_t nbr_sr_interval;

  /* Neighbors' LAN-prune-delay timer value (msecs) */
  u_int16_t nbr_pd;

  /* Neighbors' Override-Interval timer value (msecs) */
  u_int16_t nbr_oi;

  /* Neighbor hold timer thread.  */
  struct thread *t_holdtimer;
  /* Join/Prune Message bulking*/
  struct stream *jp_bulk_message;
```

```
  /* Have to update the no_of_groups while sending.*/
  u_int8_t *jp_header_group_count;
  u_int8_t group_count;

};
```

PIM Nexthop Structure

PIM nexthop table is maintained to keep track of the RPF (Reverse Path Forwarding) to reach to the multicast source as well as the PIM neighbors.

Definition

```
sstruct pim_nexthop
{
  /* Owning route node */
  struct route_node *rn;

  /* PIM VRF.  */
  struct pim_vrf *vrf;

  /* Nexthop Status Flags */
  u_int32_t flags;
#define PIM_NH_FLAG_REGISTERED  (1 << 0) /* Registered to the RIB lookup.  */
#define PIM_NH_FLAG_ACTIVE      (1 << 1) /* Active nexthop.  */
#define PIM_NH_FLAG_CHANGED     (1 << 2) /* Nexthop changed.  */
#define PIM_NH_FLAG_SRC         (1 << 3) /* Nexthop to the source.  */
#define PIM_NH_FLAG_RP          (1 << 4) /* Nexthop to the RP.  */

  /* Nexthop entries.  */
  u_int32_t nhe_size;
  struct pim_nexthop_entry *nhe_list;

  /* Mcast nexthop reference counter.  */
  u_int32_t counter;

  /* Metric associated with this next hop */
  u_int32_t metric;

  /* Preference.  */
  u_char preference;

  /* Route type of the RPF check route.  */
  u_char route_type;

  /* Prefix length of the RPF check route.  */
  u_char prefixlen;
};
```

PIM Multicast RoutingTable Structure

An MRT (Multicast Routing Table) is the union of all four types of states: (*,G), (S,G), (S,G,rpt) and (*,*,RP). The PIM MRT is the actual Tree Information Base (TIB) created for each PIM VRF instance, and is the heart of the PIM database, which generates the FCR forwarding cache entry in the forwarding plane.

Definition

```
struct pim_mrt
{
  /* Ptree node.  */
  struct ptree_node *tn;

  /* Prefix length.  This is not kept in the ptree key.  */
  u_int32_t prefixlen;

  /* PIM VRF back pointer.  */
  struct pim_vrf *vrf;

  /* MRT mode.  */
  pim_mrt_mode_t mode;

  /* MRT type.  */
  pim_mrt_type_t type;

  u_int32_t flags;
#define PIM_MRT_FLAG_MFC         (1 << 0) /* MFC entry in forwarder.  */
#define PIM_MRT_FLAG_KAT_MRIB    (1 << 1) /* MRT KAT controlled by MRIB.  */
#define PIM_MRT_FLAG_DELETED     (1 << 2) /* Deleted flag.  */
#define PIM_MRT_FLAG_DELETE_REQ  (1 << 3) /* Delete requested to MRIB.  */
#define PIM_MRT_FLAG_JT_INITIAL  (1 << 4) /* JT timer running for First time */
#define PIM_MRT_RP_CHANGE_NO_RP  (1 << 5) /* RP change event, New RP = NULL */
#define PIM_MRT_FLAG_BIDIR       (1 << 6)
#define PIM_MRT_FLAG_ECMP_REDIRECT (1 << 7) /* RPF nbr changed */

/* RPF nbr is the assert winner on the ecmp vif */
#define PIM_MRT_FLAG_ECMP_REDIRECT_RPF_NBR_ASSERT_WINNER (1 << 8)

  /* Type, Mode and address family specific methods.  */
  struct pim_mrt_methods *m;

  /* Uptime.  */
  pal_time_t uptime;

  /* Nexhtop toward source.  */
  struct pim_nexthop *nh_s;
  /* Message-id for the last message sent for this mrt */
  u_int32_t message_id;

  /* Upstream (*,*,RP), (*,G), (S,G) and (S,G,rpt) state.  */
  union
```

```
  {
    struct pim_mrt_us_state_xxrp xxrp;
    struct pim_mrt_us_state_xg xg;
    struct pim_mrt_us_state_sg sg;
    struct pim_mrt_us_state_sgrpt sgrpt;
  } us;

  /* Last RP for group G. This is stored as address so that handling
     change in RP(G) is easier.  */
  struct prefix last_rp;

  /* RPF_interface(S) or RPF_interface(RP(G)) stored to generate
   "I stops being RPF interface" event in the (S,G) and (*,G) Assert FSM.  */
  struct pim_vif *rpf_interface;

  /* Downstream VIF states. */
  u_int32_t ds_vifs_num;
  vector ds_vifs;

  /* Joins (*,*,RP) (*,G) and (S,G) VIFs. */
  mcastbitmap_t joined_olist;

  /* Inherited olist vifs
     (*,*,RP) : inherited_olist(S,G,rpt) suitably modifed for (*,*,RP)
     (*,G) : inherited_olist(S,G,rpt) suitably modifed for (*,G)
     (S,G) : inherited_olist(S,G)
     (S,G,rpt) : inherited_olist(S,G,rpt)  */
  /* For (*,G) mrt in non-patched mode, inherited_olist is per FCR */
  /* Used for BIDIR, as a temporary bitmap */
  mcastbitmap_t inherited_olist;

  /* Olist of interfaces which are in downstream (S,G,rpt) states
     of PRUNE or PRUNE TMP.  */
  mcastbitmap_t pruned_olist;

  /* Local IGMP join information.  */
  mcastbitmap_t local_olist;

  /* Mode specific entry.  */
  union
  {
#ifdef HAVE_PIM_SM
    struct pim_mrt_sm sm;
#endif /* HAVE_PIM_SM */
#ifdef HAVE_PIM_DM
    struct pim_mrt_dm dm;
#endif /* HAVE_PIM_DM */
  } um;

  /* The per-source S Forwarding Record in the (*,G) state.
```

```
     Note: This only exists in a (*,G) mrt when the forwarder does not
     support (*,G) forwarding entries. In all others mrts this is NULL.  */
  struct ptree *fcr;
  u_int32_t no_fcr;

  /* Flag for SPT switchover */
  bool_t spt_switch;  /* Only for (*,G) mrt */

#ifdef HAVE_PIM_MBR
  /* TIMER for register the DM source to SM's RP.  */
  struct thread *mbr_register_timer;

  /* Register state for register the DM source to SM's RP.  */
  u_int8_t mbr_register_state;

  /* The other pim-mode oifs for this mrt entry by mbr.  */
  mcastbitmap_t mbr_olist;

  /* Each status for mbr oifs.  */
  vector mbr_status;
#endif /* HAVE_PIM_MBR */
};
```

PIM MRT Method Table

The PIM MRT method table hides the PIM MRT type; the PIM mode as well as the address-family-specific method or function call, for example, delete PIM (S,G) MRT entry, from the rest of the PIM process. This makes the PIM core code both PIM MRT type, mode and address-family independent as possible.

Definition

```
struct pim_mrt_methods
{
  struct prefix * (*src) (const struct pim_mrt *mrt, struct prefix *src);
  struct prefix * (*grp) (const struct pim_mrt *mrt, struct prefix *grp);
  struct pim_sg_prefix * (*sg) (const struct pim_mrt *mrt,
                                struct pim_sg_prefix *sg);
  struct pim_sg_prefix * (*fcr_sg) (const struct pim_mrt_fcr *fcr,
                                    struct pim_sg_prefix *sg);
  int (*init) (struct pim_mrt *mrt);
  int (*free) (struct pim_mrt *mrt);
  int (*delete) (struct pim_mrt *mrt);
  int (*delete_success) (struct pim_mrt *mrt);
  int (*clear) (struct pim_mrt *mrt);
  int (*init_mvif) (struct pim_mrt_vif_state *mvif);
  int (*free_mvif) (struct pim_mrt_vif_state *mvif);
  struct pim_vif * (*incoming_vif) (const struct pim_mrt *mrt);
  struct pim_mrt_olist * (*get_olist) (struct pim_mrt *mrt,
                                       struct pim_vif *ivif,
                                       const mcastbitmap_t inherited_olist);
  int (*include_olist) (struct pim_mrt *mrt, mcastbitmap_t olist);
```

```
int (*exclude_olist) (struct pim_mrt *mrt, mcastbitmap_t olist);
int (*update_mfc) (struct pim_mrt *mrt);
int (*delete_mfc) (struct pim_mrt *mrt);
int (*update_mfc_by_fcr) (struct pim_mrt_fcr *fcr);
int (*delete_mfc_by_fcr) (struct pim_mrt_fcr *fcr);
int (*update_mfc_flags) (const struct pim_mrt *mrt);
int (*send_add) (struct pim_mrt *mrt, struct pim_mrt_fcr *fcr);
int (*send_delete) (struct pim_mrt *mrt, struct pim_mrt_fcr *fcr);
int (*send_stat_flags) (const struct pim_mrt *mrt,
                        const struct pim_mrt_fcr *fcr,
                        const u_int16_t flags, const u_int16_t stat_time);
int (*check_rpf_vif_changed) (struct pim_mrt *mrt);
int (*check_rpf_nbr_changed) (struct pim_mrt *mrt, struct pim_vif *vif,
                              bool_t assert_event);
int (*check_join_desired) (struct pim_mrt *mrt);
int (*check_prune_desired) (struct pim_mrt *mrt);
int (*check_register_desired) (struct pim_mrt *mrt,
                               struct pim_packet_register *reg);
int (*check_could_assert) (struct pim_mrt  *mrt, struct pim_vif *vif);
int (*check_could_register) (struct pim_mrt *mrt);
int (*check_my_assert_metric_better) (struct pim_mrt *mrt,
                                      struct pim_mrt_vif_state *mvif);
int (*check_assert_tracking_desired) (struct pim_mrt *mrt,
                                      struct pim_vif *vif);
int (*check_assert_change) (struct pim_mrt *mrt, struct pim_vif *vif);
int (*check_inherited_olist_change_fcr) (struct pim_mrt *mrt,
                                         struct pim_mrt_fcr *fcr);
int (*check_inherited_olist_change) (struct pim_mrt *mrt);
int (*check_spt_bit_rpf_change) (struct pim_mrt *mrt);
int (*process_vif_stop) (struct pim_mrt *mrt, struct pim_vif *vif);
int (*process_local_join_change) (struct pim_mrt *mrt, struct pim_vif *vif);
int (*process_local_exclude_change) (struct pim_mrt *mrt,
                                     struct pim_vif *vif);
int (*process_join_change) (struct pim_mrt *mrt,
                            struct pim_mrt_vif_state *mvif);
int (*process_prune_change) (struct pim_mrt *mrt,
                             struct pim_mrt_vif_state *mvif);
int (*process_assert_winner_change) (struct pim_mrt *mrt,
                                     struct pim_mrt_vif_state *mvif,
                                     struct prefix *old_win);
int (*process_kat_change) (struct pim_mrt *mrt);
int (*process_spt_bit_change) (struct pim_mrt *mrt);
int (*process_src_nh_change) (struct pim_mrt *mrt, struct pim_nexthop *nh);
int (*process_rp_nh_change) (struct pim_mrt *mrt, struct pim_nexthop *nh);
int (*process_nh_update) (struct pim_mrt *mrt, struct pim_nexthop *nh);
/* MBR alert methods.  */
int (*process_delete_alert) (struct pim_mrt *mrt);
int (*process_nocache_alert) (struct pim_mrt *mrt);
int (*process_join_alert) (struct pim_mrt *mrt, struct pim_vif *vif);
int (*process_prune_alert) (struct pim_mrt *mrt, struct pim_vif *vif);
```

```
};
```

PIM RP Structure

The sections shows the data structure for the PIM - RP.

Definition

```
struct pim_rp
{
  struct pim_rp *next;
  struct pim_rp *prev;

  /* PIM VRF back pointer.  */
  struct pim_vrf *vrf;

  /* Address of RP.  */
  struct prefix addr;

  /* From. */
  struct prefix from;

  /* PIM Mode */
  pim_mode mode;

  /* Type of RP entry.  */
  u_char type;
#define PIM_RP_TYPE_STATIC                    (1)
#define PIM_RP_TYPE_BSR                       (2)
#ifdef HAVE_PIM_IPV6
#define PIM_RP_TYPE_EMBED                     (3)
#endif /* HAVE_PIM_IPV6 */

  /* Flags.  */
  u_char flags;
#define PIM_RP_FLAG_SELF                      (1 << 0)
#define PIM_RP_FLAG_UNUPDATED                 (1 << 1)
#define PIM_RP_FLAG_ACTIVE                    (1 << 2)
#define PIM_RP_FLAG_STATIC_OVRD               (1 << 3)
#define PIM_RP_FLAG_SNMP_STATIC               (1 << 4)
#define PIM_RP_FLAG_BIDIR                     (1 << 5)

  /* Priority.  */
  u_char priority;

  /* Holdtime. */
  u_int16_t holdtime;

  /* Timer.  */
  struct thread *t_holdtime;
  /* Back pointer to the RP set.  */
```

```
  struct pim_rp_set *rp_set;

#ifdef HAVE_PIM_IPV6
  /* Back pointer to the P-Trie node for embedded RP.  */
  void *pn;
#endif /* HAVE_PIM_IPV6 */

  /* Pointer to static RP configuration.  */
  struct pim_rp_static_conf *st_rp_conf;

  /* Uptime and expire time. */
  pal_time_t uptime;
  pal_time_t exptime;

#ifdef HAVE_PIM_IPV6
    /* Reference count for embedded RP */
  u_int32_t ref_cnt;
#endif
};
```

Multicast Source Discovery Protocol Structure

This section shows the data structure for Multicast Source Discovery Protocol (MSDP).

Definition

```
struct msdp
{
  /* Back pointer to the MSDP VRF.  */
  MSDP_VRF *vrf;

  /* List of MSDP peers */
  struct list *peer_list;

  /* Mesh groups. Lookups will be easier. */
  struct list *mesh_group_list;

  u_int32_t cflags;
#define MSDP_ORIGINATOR_ID (1 << 0)

  /* Originator-id. */
  char *ifname;

  /* SA-CACHE */
  struct ptree *sa_cache;

  /* Place holder for rp nodes in the cache ptree.
     Faster processing during cache re-advertisement. */
  struct list *sa_cache_rp_list;

  struct thread *t_sa_adv_timer;
```

```
#define MSDP_SA_ADV_PERIOD 60

  /* MSDP Server (Listen) Socket Threads List */
  struct msdp_listen_sock_lnode *listen_sock_lnode;
};
```

Messages

MRIBD Messages

The sections that follow show the data structures for the PIM - MRIBD IPC messages.

mrib4_msg_igmp

This message is sent by MRIBD to notify PIM about an IGMP join/leave message.

```
struct mrib4_msg_igmp
{
  /* Ifindex */
  u_int32_t ifindex;

  /* Filter Mode (INCLUDE | EXCLUDE) */
  u_int16_t filt_mode;
#define MRIB4_MSG_IGMP_FILT_MODE_INCL             0
#define MRIB4_MSG_IGMP_FILT_MODE_EXCL             1

  /* Number of Sources */
  u_int16_t num_srcs;

  /* Group address */
  struct pal_in4_addr grp_addr;

  /* Prefix length: default 32 */
  u_int8_t prefixlen;

  /* Source addresses List */
  struct pal_in4_addr src_addr_list [1];
};
```

mrib6_msg_mld

This message is sent by MRIBD to notify PIM about MLD join/leave related information.

```
struct mrib6_msg_mld
{
  /* Ifindex */
  u_int32_t ifindex;

  /* Filter Mode (INCLUDE | EXCLUDE) */
  u_int16_t filt_mode;
#define MRIB6_MSG_MLD_FILT_MODE_INCL              0
```

```
#define MRIB6_MSG_MLD_FILT_MODE_EXCL              1

  /* Number of Sources */
  u_int16_t num_srcs;

  /* Group address */
  struct pal_in6_addr grp_addr;

  /* Source addresses List */
  struct pal_in6_addr src_addr_list [1];
};
```

BGP Messages

This section shows the data structures that PIM and BGP use to communicate. Multicast Source Discovery Protocol (MSDP) depends on this information for peer-RPF checks.

bgp_msg_prefix_lookup_ipv4

PIM send this message to BGP to request an AS number and BGP route.

```
struct bgp_msg_prefix_lookup_ipv4
{
  struct pal_in4_addr addr;

  u_char prefixlen;
};
#define  BGP_MSG_IPV4_PREFIX_AS_SIZE      5
```

bgp_msg_ipv4_prefix_as

BGP sends this message to PIM in reply to an AS number request.

```
struct bgp_msg_ipv4_prefix_as
{
  struct pal_in4_addr addr;

  u_int32_t as_no;
};
#define  BGP_MSG_IPV4_AS_SIZE  8
```

bgp_msg_ipv4_route

BGP sends this message to PIM in reply to a BGP route request.

```
struct bgp_msg_ipv4_route
{
  struct pal_in4_addr prefix;
  u_char prefixlen;

  struct pal_in4_addr peer_addr;

  u_int32_t ifindex;
  struct pal_in4_addr nexthop_addr;
```

```
  u_int32_t as_count;
  u_int8_t*   as_path;
};
#define BGP_MSG_IPV4_ROUTE_SIZE   20  /* Minimum size */
```

CHAPTER 4 PIM4 Command API

The functions in this chapter are called by the PIM IPv4 commands

msdp_api_default_peer_set

This function sets an Multicast Source Discovery Protocol (MSDP) peer from which to accept Source-Active (SA) messages.

The `ip msdp default-peer` command calls this function.

Syntax

```
int
msdp_api_default_peer_set (u_int32_t vr_id, char *vrf_name,
                           struct pal_in4_addr peer_addr, u_int8_t *alist)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer
`alist`	Make this the default peer only for this access list number of rendezvous points (RPs)

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERR_PEER_ALREADY_DEFAULT when a peer with the address `peer_addr` is already the default

MSDP_API_SET_ERR_PEER_DOESNT_EXIST when a peer with the address `peer_addr` does not exist

msdp_api_default_peer_unset

This function ends accepting SA messages from a Multicast Source Discovery Protocol (MSDP) peer.

The `no ip msdp default-peer` command calls this function.

Syntax

```
int
msdp_api_default_peer_unset (u_int32_t vr_id, char *vrf_name,
                             struct pal_in4_addr peer_addr)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERR_DEFAULT_PEER_NOT_CONFIGURED when a peer with the address `peer_addr` is not the default

MSDP_API_SET_ERR_PEER_DOESNT_EXIST when a peer with the address `peer_addr` does not exist

msdp_api_mesh_group_set

This function adds a Multicast Source Discovery Protocol (MSDP) peer to a mesh group.

You can set up multiple mesh groups on the same device and multiple peers per mesh group.

The `ip msdp mesh-group` command calls this function.

Syntax

```
int
msdp_api_mesh_group_set (u_int32_t vr_id, char *vrf_name,
                         struct pal_in4_addr peer_addr, char *mesh_grp_name)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer
`mesh_grp_name`	Name of the mesh group

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERROR when `vrf_name` is not valid or there is an internal error

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERR_PEER_ALREADY_IN_THE_MESH_GROUP when a peer with the address `peer_addr` is already part of `mesh_grp_name`

MSDP_API_SET_ERR_MEM_ALLOC_FAIL when memory allocation fails

MSDP_API_SET_ERR_PEER_DOESNT_EXIST when a peer with the address `peer_addr` does not exist

msdp_api_mesh_group_unset

This function removes a Multicast Source Discovery Protocol (MSDP) peer from a mesh group.

The `no ip msdp mesh-group` command calls this function.

Syntax

```
int
msdp_api_mesh_group_unset (u_int32_t vr_id, char *vrf_name,
                           struct pal_in4_addr peer_addr, char *mesh_grp_name)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer
`mesh_grp_name`	Name of the mesh group

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERR_PEER_NOT_IN_A_MESH_GROUP when a peer with the address `peer_addr` is not found in `mesh_grp_name`

MSDP_API_SET_ERR_MESH_GROUP_DOESNT_EXIST when `mesh_grp_name` does not exist

MSDP_API_SET_ERR_PEER_DOESNT_EXIST when a peer with the address `peer_addr` does not exist

msdp_api_msdp_peer_clear

This function clears the TCP connection to a Multicast Source Discovery Protocol (MSDP) peer.

This function closes the TCP connection to the peer, resets all the MSDP peer statistics, and clears the input and output queues to and from the MSDP peer.

The `clear ip msdp peer` command calls this function.

Syntax

```
int
msdp_api_msdp_peer_clear (u_int32_t vr_id, char *vrf_name, struct pal_in4_addr
```

```
                          *peer_addr)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

msdp_api_msdp_peer_show

This function displays information about a Multicast Source Discovery Protocol (MSDP) peer.

The `show ip msdp peer` command calls this function.

Syntax

```
int
msdp_api_msdp_peer_show (struct cli *cli, u_int32_t vr_id, char *vrf_name,
                         struct pal_in4_addr *peer_addr)
```

Input Parameters

`cli`	CLI structure
`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

msdp_api_originator_id_set

This function allows a Multicast Source Discovery Protocol (MSDP) speaker that originates a Source-Active (SA) message to use the IP address of an interface as a rendezvous point (RP) address in the SA message.

By default, ZebOS-XP uses the RP address of the device.

The `ip msdp originator-id` command calls this function.

Syntax

```
int
msdp_api_originator_id_set (u_int32_t vr_id, char *vrf_name, char *ifname)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`ifname`	Use the IP address of this interface as an RP address in SA messages

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_ORIGINATOR_ID_ALREADY_CONFIGURED when the address of `ifname` is already set as an RP

msdp_api_originator_id_unset

This function specifies to use the rendezvous point (RP) of the device in Source-Active (SA) messages for Multicast Source Discovery Protocol (MSDP).

The `no ip msdp originator-id` command calls this function.

Syntax

```
int
msdp_api_originator_id_unset (u_int32_t vr_id, char *vrf_name, char *ifname)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`ifname`	Interface whose IP address is used as an RP address in SA messages

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_ORIGINATOR_MISMATCH when the address of `ifname` is not set as an RP

MSDP_API_SET_ERR_ORIGINATOR_NOT_CONFIGURED when an address is not set as an RP

msdp_api_peer_password_set

This function sets an MD5-shared password key used for authenticating a Multicast Source Discovery Protocol (MSDP) peer. By default, no MD5 password is enabled.

The `ip msdp password` command calls this function.

Syntax

```
int
msdp_api_peer_password_set (u_int32_t vr_id, char *vrf_name,
                            struct pal_in4_addr peer_addr, u_int8_t type,
                            u_int8_t *password)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer
`type`	Reserved for future use
`password`	Password (maximum 80 characters)

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when `vrf_name` does not exist

MSDP_API_SET_ERR_PASSWORD_ALREADY_CONF when `password` is already the password

MSDP_API_SET_ERR_PASSWORD_LENGTH_EXCEEDED when `password` exceeds the maximum length

MSDP_API_SET_ERR_MEM_ALLOC_FAIL when memory allocation fails

msdp_api_peer_password_unset

This function removes an MD5-shared password key used for authenticating a Multicast Source Discovery Protocol (MSDP) peer.

The `no ip msdp password` command calls this function.

Syntax

```
int
msdp_api_peer_password_unset (u_int32_t vr_id, char *vrf_name,
                              struct pal_in4_addr peer_addr, u_int8_t type,
                              u_int8_t *password)
```

Input Parameters

vr_id	Virtual router ID
vrf_name	VPN routing/forwarding name
peer_addr	IPv4 address of peer
type	Reserved for future use
password	Password (maximum 80 characters)

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when vr_id does not exist

MSDP_API_SET_ERR_WRONG_VRF when vrf_name does not exist

MSDP_API_SET_ERR_PASSWORD_NOT_CONF when a password is not set

msdp_api_peer_set

This function configures a Multicast Source Discovery Protocol (MSDP) peer relationship.

The ip msdp peer command calls this function.

Syntax

```
int
msdp_api_peer_set (u_int32_t vr_id, char *vrf_name,
                   char *ifname, struct pal_in4_addr peer_addr)
```

Input Parameters

vr_id	Virtual router ID
vrf_name	VPN routing/forwarding name
ifname	Use the primary address of this interface for the TCP connection with the peer
peer_addr	IPv4 address of peer

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when vr_id does not exist

MSDP_API_SET_ERR_WRONG_VRF when vrf_name does not exist

MSDP_API_SET_ERROR when vrf_name is not valid

MSDP_API_SET_ERR_PEER_ALREADY_EXISTS when there is already a relationship with a peer having the address peer_addr

MSDP_API_SET_ERR_MEM_ALLOC_FAIL when memory allocation fails

msdp_api_peer_unset

This function removes a Multicast Source Discovery Protocol (MSDP) peer relationship.

The `no ip msdp peer` command calls this function.

Syntax

```
int
msdp_api_peer_unset (u_int32_t vr_id, char *vrf_name,
                     struct pal_in4_addr peer_addr)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`peer_addr`	IPv4 address of peer

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when the `vrf_name` does not exist

MSDP_API_SET_ERROR when `vrf_name` is not valid

MSDP_API_SET_ERR_PEER_DOESNT_EXIST when there is not a relationship with a peer having the address `peer_addr`

msdp_api_sa_cache_clear

This function clears Multicast Source Discovery Protocol (MSDP) Source-Active (SA) cache entries.

The `clear ip msdp sa-cache` command calls this function.

Syntax

```
int
msdp_api_sa_cache_clear (u_int32_t vr_id, char *vrf_name,
                         bool_t grp_addr_provided, struct pal_in4_addr grp_addr)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_name`	VPN routing/forwarding name
`grp_addr_provided`	Specify `PAL_TRUE` if a group address is specified in `grp_addr` or `PAL_FALSE` to clear all SA cache entries
`grp_addr`	Multicast group address

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when the `vrf_name` does not exist

msdp_api_sa_cache_show

This function displays the (S,G) state learned from Multicast Source Discovery Protocol (MSDP) peers.

The addresses in `src` and `grp` are optional:

- If you specify `NULL` for both addresses, the entire Source-Active (SA) cache is displayed
- If you specify `grp` and `NULL` for `src`, all sources for that group address are displayed
- If you specify `src` and `NULL` for `grp`, all groups for that source address are displayed
- If you specify both `src` and `grp`, the (S, G) entry for those addresses is displayed

The `show ip msdp sa-cache` command calls this function.

Syntax

```
int
msdp_api_sa_cache_show (struct cli *cli, u_int32_t vr_id, char *name,
                        struct pal_in4_addr *src, struct pal_in4_addr *grp)
```

Input Parameters

`cli`	CLI structure
`vr_id`	Virtual router ID
`name`	VPN routing/forwarding name
`src`	Source address
`grp`	Group address

Output Parameters

None

Return Values

MSDP_API_SET_SUCCESS when the function succeeds

MSDP_API_SET_ERR_VR_DOESNT_EXIST when `vr_id` does not exist

MSDP_API_SET_ERR_WRONG_VRF when the `vrf_name` does not exist

pim4_api_anycast_rp_set

This function configures an anycast rendezvous point.

Syntax

```
int
pim4_api_anycast_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                         struct pal_in4_addr *anycast_rp_addr,
                         struct pal_in4_addr *member_rp_addr);
```

Input Parameters

`vr_id` Virtual router ID

`vrf_id` VPN routing/forwarding instance ID

Output Parameters

`anycast_rp_addr`

Address of the anycast RP

`member_rp_addr`

Communication IP address between the configured RPs in the RP set

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF is not found

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_RP_ANYCAST_SHOULD_BE_UNICAST

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_anycast_rp_unset

This function removes configuration of an anycast RP

Syntax

```
int
pim4_api_anycast_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                           struct pal_in4_addr *anycast_rp_addr,
                           struct pal_in4_addr *member_rp_addr);
```

Input Parameters

`vr_id` Virtual router ID

`vrf_id` VPN routing/forwarding instance ID

Output Parameters

`anycast_rp_addr`

Address of the anycast RP

`member_rp_addr`

Communication IP address between the configured RPs in the RP set

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF is not found

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_RP_ANYCAST_SHOULD_BE_UNICAST

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bidir_disable

This function disables bidirectional PIM.

Syntax

```
int
pim4_api_bidir_disable (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID(0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_bidir_enable

This function enables bidirectional PIM.

Syntax

```
int
pim4_api_bidir_enable (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID(0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_bsr_candidate_set

This function sets the specified router as candidate BSR using the interface name as its address.

Syntax

```
int
pim4_api_bsr_candidate_set (u_int32_t vr_id, vrf_id_t vrf_id,
                            char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	The name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_candidate_unset

This function removes the specified router as candidate BSR.

Syntax

```
int
pim4_api_bsr_candidate_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                              char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface.

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_candidate_hash_mask_set

This function configures the hash-mask length for a candidate BSR.

Syntax

```
int
pim4_api_bsr_candidate_hash_mask_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      char *ifname, u_char hash_mask);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`hash_mask`	Hash mask length used to hash for RPs

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_candidate_hash_mask_unset

This function removes configuration of a hash-mask length for a candidate BSR.

Syntax

```
int
pim4_api_bsr_candidate_hash_mask_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_candidate_priority_set

This function configures the priority value for a candidate BSR.

Syntax

```
int
pim4_api_bsr_candidate_priority_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     char *ifname, u_char priority);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`priority`	Priority value assigned to the BSR

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_candidate_priority_unset

This function removes the configuration of a priority value for a candidate BSR.

Syntax

```
int
pim4_api_bsr_candidate_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_interop_set

This function configures bootstrap router interoperability.

Syntax

```
int
pim4_api_bsr_interop_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_bsr_interop_unset

This function removes configuration of BSR interoperability,.

Syntax

```
int
pim4_api_bsr_interop_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_clear_bsr_rpset

This function clears the specified router as the candidate BSR RP set.

Syntax

```
int
pim4_api_clear_bsr_rpset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR when BSR information is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_clear_tib

This function is clears all TIB states and MFC cache entries.

Syntax

```
int
pim4_api_clear_tib (u_int32_t vr_id, vrf_id_t vrf_id, pim_api_mode_t mode,
                    struct pal_in4_addr *src, struct pal_in4_addr *grp);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`mode`	Mode of operation, either dense more or sparse mode

Output Parameters

`src`	Source IP address
`grp`	Group IP address

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_crp_per_grp_chk

This function configures a group range for CRP.

Syntax

```
int
pim4_api_crp_per_grp_chk (u_int32_t vr_id, vrf_id_t vrf_id,
                          char *ifname, char *group_acl_name)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`*group_acl_name`	Name of the ACL to use a the group range.

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_CHECK_ERR_CRP_IF_OR_GROUP_IN_USED

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_all_set

This function enables debugging for all PIM events.

Syntax

```
int
pim4_api_debug_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_all_unset

This function disables debugging for all PIM events.

Syntax

```
int
pim4_api_debug_all_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_event_set

This function enables debugging of a PIM event.

Syntax

```
int
pim4_api_debug_event_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_event_unset

This function disables debugging of a PIM event.

Syntax

```
int
pim4_api_debug_event_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mfc_set

This function enables debugging for MFC updates.

Syntax

```
int
pim4_api_debug_mfc_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mfc_unset

This function disables debugging of MFC updates.

Syntax

```
int
pim4_api_debug_mfc_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mib_set

This function enables debugging of MIB entries.

Syntax

```
int
pim4_api_debug_mib_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mib_unset

This function disables debugging of MIB entries.

Syntax

```
int
pim4_api_debug_mib_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mtrace_set

This function enables debugging of MTRACE messages.

Syntax

```
int
pim4_api_debug_mtrace_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_mtrace_unset

This function disables debugging of MTRACE messages.

Syntax

```
int
pim4_api_debug_mtrace_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_nexthop_set

This function enables debugging of Reverse Path Forwarding (RPF) nexthop cache handling.

Syntax

```
int
pim4_api_debug_nexthop_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_nexthop_unset

This function disables debugging of RPF nexthop cache handing.

Syntax

```
int
pim4_api_debug_nexthop_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_nsm_set

This function enables debugging of NSM events.

Syntax

```
int
pim4_api_debug_nsm_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_nsm_unset

This function disables debugging of NSM events.

Syntax

```
int
pim4_api_debug_nsm_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_all_set

This function enables debugging of all PIM packets.

Syntax

```
int
pim4_api_debug_packet_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_all_unset

This function disables debugging of all PIM packets.

Syntax

```
int
pim4_api_debug_packet_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_in_set

This function enables debugging of an incoming PIM packet.

Syntax

```
int
pim4_api_debug_packet_in_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_in_unset

This function disables debugging of an incoming PIM packet.

Syntax

```
int
pim4_api_debug_packet_in_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_out_set

This function enables debugging of an outgoing PIM packet.

Syntax

```
int
pim4_api_debug_packet_out_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_packet_out_unset

This function disables debugging of an outgoing PIM packet.

Syntax

```
int
pim4_api_debug_packet_out_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_state_set

This function enables debugging of the PIM state.

Syntax

```
int
pim4_api_debug_state_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_state_unset

This function disables debugging of the PIM state.

Syntax

```
int
pim4_api_debug_state_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_all_set

This function enables debugging of timers.

Syntax

```
int
pim4_api_debug_timer_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_all_unset

This function disables debugging of timers.

Syntax

```
int
pim4_api_debug_timer_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_assert_all_set

This function enables debugging of all Assert timer.s

Syntax

```
int
```

```
pim4_api_debug_timer_assert_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_assert_all_unset

This function disables debugging of all Assert timers.

Syntax

```
int
pim4_api_debug_timer_assert_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_assert_timer_set

This function enables debugging of the PIM assert timer.

Syntax

```
int
pim4_api_debug_timer_assert_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
```

```
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_assert_timer_unset

This function stops debugging the debugging of the PIM assert timer.

Syntax

```
int
pim4_api_debug_timer_assert_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                         int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_all_set

This function enables debugging of the Bootstrap Router timers.

Syntax

```
int
pim4_api_debug_timer_bsr_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                  int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_all_unset

This function disables debugging of the BSR timers.

Syntax

```
int
pim4_api_debug_timer_bsr_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_bootstrap_set

This function enables debugging of the BSR timer.

Syntax

```
int
pim4_api_debug_timer_bsr_bootstrap_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_bootstrap_unset

This function disables debugging of the BSR timer.

Syntax

```
int
pim4_api_debug_timer_bsr_bootstrap_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_candidate_rp_set

This function enables debugging of the BSR Candidate-RP timer.

Syntax

```
int
pim4_api_debug_timer_bsr_candidate_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                           int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_bsr_bootstrap_unset

This function disables debugging of the BSR Candidate-RP timer.

Syntax

```
int
pim4_api_debug_timer_bsr_candidate_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                             int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_all_set

This function enables debugging of the hello timers.

Syntax

```
int
pim4_api_debug_timer_hello_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_all_unset

This function disables debugging of hello timers,

Syntax

```
int
pim4_api_debug_timer_hello_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_neighbor_liveliness_set

This function disables debugging of the neighbor-liveliness hello timer.

Syntax

```
int
pim4_api_debug_timer_hello_neighbor_liveliness_set (u_int32_t vr_id,
                                                    vrf_id_t vrf_id,
                                                    int cli_mode)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_neighbor_liveliness_unset

This function disables debugging of the neighbor-liveliness hello timer.

Syntax

```
int
pim4_api_debug_timer_hello_neighbor_liveliness_unset (u_int32_t vr_id,
                                                      vrf_id_t vrf_id,
                                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_timer_set

This function sets the debugging of hello timers.

Syntax

```
int
pim4_api_debug_timer_hello_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_timer_unset

This function disables the debugging of hello timers.

Syntax

```
int
pim4_api_debug_timer_hello_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_triggered_set

This function enables debugging of the triggered-hello timer.

Syntax

```
int
pim4_api_debug_timer_hello_triggered_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_hello_triggered_unset

This function disables debugging of the triggered-hello timer.

Syntax

```
int
pim4_api_debug_timer_hello_triggered_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                            int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_all_set

This function enables debugging of all Join/Prune timers.

Syntax

```
int
pim4_api_debug_timer_jp_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_all_unset

This function disables debugging of all Join/Prune timers.

Syntax

```
int
pim4_api_debug_timer_jp_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                   int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_expiry_set

This function enables debugging of the join/prune expiration timer.

Syntax

```
int
pim4_api_debug_timer_jp_expiry_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_expiry_unset

This function disables debugging of the join/prune expiration timer.

Syntax

```
int
pim4_api_debug_timer_jp_expiry_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_keep_alive_set

This function enables debugging of the join/prune keepalive timer.

Syntax

```
int
pim4_api_debug_timer_jp_keep_alive_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	Command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_keep_alive_unset

This function disables debugging oft he join/prune keepalive timer.

Syntax

```
int
pim4_api_debug_timer_jp_keep_alive_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_override_set

This function enables debugging of the join/prune upstream override timer.

Syntax

```
int
pim4_api_debug_timer_jp_override_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_override_unset

This function disables debugging of the join/prune upstream override timer.

Syntax

```
int
pim4_api_debug_timer_jp_override_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_prune_pending_set

This function enables debugging of the join/prune pending timer.

Syntax

```
    int
pim4_api_debug_timer_jp_prune_pending_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                           int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_prune_pending_unset

This function disables debugging of the join/prune pending timer.

Syntax

```
int
pim4_api_debug_timer_jp_prune_pending_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                             int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_timer_set

This function enables debugging of the join/prune timer.

Syntax

```
int
pim4_api_debug_timer_jp_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                   int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_jp_timer_unset

This function disables debugging of the join/prune timer.

Syntax

```
int
pim4_api_debug_timer_jp_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                     int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_register_all_set

This function enables debugging of the Register timers.

Syntax

```
int
pim4_api_debug_timer_register_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                       int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_register_all_unset

This function disables debugging of the Register timers.

Syntax

```
int
pim4_api_debug_timer_register_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                         int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_register_stop_set

This function enables the debugging of the Register Stop timer.

Syntax

```
int
pim4_api_debug_timer_register_stop_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_debug_timer_register_stop_unset

This function disables debugging of the Register Stop timer.

Syntax

```
int
pim4_api_debug_timer_register_stop_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_bidir_nbr_filter_set

This function configures the PIM VIF bidirectional neighbor filter access-list name

Syntax

```
int
pim4_api_vif_bidir_nbr_filter_set (u_int32_t vr_id, char *ifname,
                                   char *filter);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_df_offer_interval_set

This function configures the interval for a Designated Forwarder (DF) Offer Message.

Syntax

```
int
pim4_api_df_offer_interval_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer interval

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_df_offer_interval_unset

This function unconfigures the interval for a Designated Forwarder (DF) Offer Message.

Syntax

```
int
pim4_api_df_offer_interval_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                  u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer interval

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_df_offer_limit_set

This function configures the limit for a Designated Forwarder (DF) Offer Message

Syntax

```
int
pim4_api_df_offer_limit_set (u_int32_t vr_id, vrf_id_t vrf_id,
                             u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer limit

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_df_offer_limit_unset

This function unconfigures the limit for a Designated Forwarder (DF) Offer Message

Syntax

```
int
pim4_api_df_offer_limit_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                               u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)

`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer limit

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_dm_group_default_set

This function sets a default dense-group address

Syntax

```
int
pim4_api_dm_group_default_set (u_int32_t vr_id, vrf_id_t vrf_id,
                           struct pal_in4_addr *grp_addr)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`grp_addr`	IP address of dense-group

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

PIM_API_DM_MEMBER_ALREADY_PRESENT when the group address has already been configured

pim4_api_dm_group_default_unset

This function unsets a default dense-group address

Syntax

```
int
pim4_api_dm_group_default_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                           struct pal_in4_addr *grp_addr)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`grp_addr`	IP address of dense-group

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

PIM_API_DM_MEMBER_ALREADY_PRESENT when the group address has already been configured

pim4_api_ecmp_bundle_create

This function creates an ECMP bundle.

Syntax

```
int
pim4_api_ecmp_bundle_create (u_int32_t vr_id, vrf_id_t vrf_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM VR is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_NAME_TOO_LONG when the bundle name exceeds the limit of 50 characters.

PIM_API_SET_ERR_OUT_OF_MEMORY when the process has run out of memory

PIM_API_SET_ERR_ECMP_BUNDLE_EXISTS when the bundle already exists

PIM_API_SUCCESS when the function executes proper

pim4_api_ecmp_bundle_delete

This function deletes an ECMP bundle.

Syntax

```
int
```

```
pim4_api_ecmp_bundle_delete (u_int32_t vr_id, vrf_id_t vrf_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM VR is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ECMP_BUNDLE_NOT_FOUND when the specified bundle cannot be found

PIM_API_SUCCESS when the function executes properly

pim4_api_ignore_rp_set_priority_set

This function configures PIM to ignore the RP priority while performing RP selection.

Syntax

```
int
pim4_api_ignore_rp_set_priority_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_ignore_rp_set_priority_unset

This function removes the configuration for ignoring the RP priority with electing an RP.

Syntax

```
int
pim4_api_ignore_rp_set_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_join_prune_timer_set

This function configures a PIM join/prune timer and set its value.

Syntax

```
int
pim4_api_join_prune_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                               u_int32_t timer);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`timer`	Join/prune timer value, in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_join_prune_timer_unset

This function removes the configuration of a join/prune timer.

Syntax

```
int
pim4_api_join_prune_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_source_address_set

This function configures the source address of Register messages.

Syntax

```
int
pim4_api_register_source_address_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      struct pal_in4_addr *soruce_addr);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`source_addr`	IP address to use as the source address for Registers

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_source_interface_set

This function sets the source address of Register messages to the address of the given interface.

Syntax

```
int
pim4_api_register_source_interface_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Register source interface name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_source_unset

This function removes configuration of a source address for Register messages.

Syntax

```
int
pim4_api_register_source_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_rate_limit_set

This function configures the maximum number of Registers to generate for an (S,G).

Syntax

```
int
pim4_api_register_rate_limit_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                  u_int16_t rate_limit);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`rate_limit`	Maximum number of Register messages allowed, in packets per second

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_rate_limit_unset

This function removes the configuration of a maximum number of register messages allowed for an (S,G).

Syntax

```
int
pim4_api_register_rate_limit_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_rp_reachability_check_set

This function configures an RP unicast reachability check in the Register state machine.

Syntax

```
int
pim4_api_register_rp_reachability_check_set (u_int32_t vr_id,
                                             vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_rp_reachability_check_unset

This function removes configuration of an RP unicast reachability check in the Register state machine.

Syntax

```
int
pim4_api_register_rp_reachability_check_unset (u_int32_t vr_id,
                                               vrf_id_t vrf_id);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_set_all

This function configures an RP candidate.

Syntax

```
int
pim4_api_rp_candidate_set_all (u_int32_t vr_id, vrf_id_t vrf_id,
                               char *ifname, bool_t is_bidir, char *acl,
                               s_int32_t interval,s_int32_t priority);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Interface name
is_bidir	Whether bidirectional
acl	Access-list name
interval	Interval value of RP candidate
priority	Priority value of RP candidate

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_register_keep_alive_timer_set

This function configures the keepalive timer (KAT) value of (S,G) created by the RP by register messages.

Syntax

```
int
pim4_api_rp_register_keep_alive_timer_set (u_int32_t vr_id,
                                           vrf_id_t vrf_id,
                                           u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`sec`	Keepalive-timer value, in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_register_keep_alive_timer_unset

This function removes configuration of the KAT value of (S,G) created at the RP by Register messages. The KAT value is then reset to (Register Suppression Time * 3) + Register Probe Interval.

Syntax

```
int
pim4_api_rp_register_keep_alive_timer_unset (u_int32_t vr_id,
                                             vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_suppression_time_set

This function configures the Register Suppression interval.

Syntax

```
int
pim4_api_register_suppression_time_set (u_int32_t vr_id,
                                        vrf_id_t vrf_id,
                                        u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`sec`	Suppression timer value, in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_register_suppression_time_unset

This function removes the configuration of a register suppression interval.

Syntax

```
int
pim4_api_register_suppression_time_unset (u_int32_t vr_id,
                                          vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_accept_register_filter_set

This function configures an ACL to filter sources allowed to register with this RP.

Syntax

```
int
pim4_api_rp_accept_register_filter_set (u_int32_t vr_id,
                                        vrf_id_t vrf_id,
                                        char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_accept_register_filter_unset

This function removes the configuration of an ACL to filter sources allowed to register with this RP.

Syntax

```
int
pim4_api_rp_accept_register_filter_unset (u_int32_t vr_id,
                                          vrf_id_t vrf_id,
                                          char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_set

This function configures an RP candidate set and sets its priority.

Syntax

```
int
pim4_api_rp_candidate_set (u_int32_t vr_id, vrf_id_t vrf_id,
                           char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_BIDIR_DISABLED when Bidirectional forwarding is not enabled

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_unset

This function removes configuration of a candidate RP set.

Syntax

```
int
pim4_api_rp_candidate_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                             char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_BIDIR_DISABLED when bidirectional forwarding is not enabled.

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_priority_set

This function configures the priority of an RP candidate set.

Syntax

```
int
pim4_api_rp_candidate_priority_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    char *ifname, u_char priority);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface
priority	Priority value of the RP candidate set

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_priority_unset

This function removes the configuration of a priority for an RP candidate set.

Syntax

```
int
pim4_api_rp_candidate_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      char *ifname);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_checksum_filter_set

This function configures a Cisco-style Register checksum. The filter ACL controls the groups to which the checksum calculation applies.

Syntax

```
int
pim4_api_rp_checksum_filter_set (u_int32_t vr_id,
                                 vrf_id_t vrf_id,
                                 char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_checksum_filter_unset

This function removes configuration of a Cisco-style Register checksum.

Syntax

```
int
pim4_api_rp_checksum_filter_unset (u_int32_t vr_id,
                                   vrf_id_t vrf_id,
                                   char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_adv_interval_set

This function configures a candidate RP advertisement interval.

Syntax

```
int
pim4_api_rp_candidate_adv_interval_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname, u_int16_t sec);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface
sec	The C-RP advertisement interval, in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_adv_interval_unset

This function removes a candidate RP advertisement interval.

Syntax

```
int
pim4_api_rp_candidate_adv_interval_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          char *ifname);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_group_acl_set

This function configures a group range using an access control list for a candidate RP set.

Syntax

```
int
pim4_api_rp_candidate_group_acl_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     char *ifname, char *group_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`group_acl_name`	Name of the ACL to use a the group range

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_rp_candidate_group_acl_unset

This function removes configuration of a group range for a candidate RP set.

Syntax

```
int
pim4_api_rp_candidate_group_acl_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_router_id_set

This function sets a router-id.

Syntax

```
int
pim4_api_router_id_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        struct pal_in4_addr router_id)
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
router_id	Router-ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when the interface name does not exists

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ROUTER_ID_INVALID when the router-id cannot be configured with the value

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_router_id_unset

This function deletes the router-id.

Syntax

```
int
pim4_api_router_id_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          struct pal_in4_addr router_id)
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
router_id	Router-ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when the interface name does not exists

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ROUTER_ID_INVALID when the router-id cannot be configured with the value

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_static_rp_set

This function configures a static rendezvous point.

Syntax

```
int
pim4_api_static_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        struct pal_in4_addr *rp_addr,
                        char *acl_name, bool_t override_flag);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`acl_name`	Name of ACL (access control list) to use.
`override_flag`	Override flag.

Output Parameters

`rp_addr`	Rendezvous point address

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_static_rp_unset

This function removes configuration of a static rendezvous point.

Syntax

```
int
pim4_api_static_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          struct pal_in4_addr *rp_addr);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

`rp_addr`	Rendezvous point address

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_spt_switch_threshold_set

This function configures an SPT (system posture token) switchover threshold. The group list ACL filters groups for which SPT switchover is performed.

Syntax

```
int
pim4_api_spt_switch_threshold_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                   char *group_list_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`group_list_acl_name`	Name of the ACL to use as the group range

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_spt_switch_threshold_unset

This function removes the configuration of an SPT switchover threshold.

Syntax

```
int
pim4_api_spt_switch_threshold_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                     char *group_list_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`group_list_acl_name`	Name of the ACL used as the group range

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_ssm_default_set

This function configures PIM SSM operation.

Syntax

```
int
pim4_api_ssm_default_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_ssm_default_unset

This function removes the configuration for PIM SSM operation.

Syntax

```
int
pim4_api_ssm_default_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_ssm_range_set

This function configures an SSM range using an access control list.

Syntax

```
int
pim4_api_ssm_range_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        char *acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`acl_name`	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_ssm_range_unset

This function removes configuration of an SSM range using an ACL.

Syntax

```
int
pim4_api_ssm_range_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          char *acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`acl_name`	ACL name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_bidir_nbr_filter_unset

This function unconfigures the PIM VIF bidirectional neighbor filter access-list name

Syntax

```
int
pim4_api_vif_bidir_nbr_filter_unset (u_int32_t vr_id, char *ifname,
                                     char *filter);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_vif_bind_bundle

This function binds the interface to an ECMP Bundle.

Syntax

```
int
pim4_api_vif_bind_bundle (u_int32_t vr_id, char *ifname, char *bundle_name)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`ifname`	Interface Name
`bundle_name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when the interface name does not exists

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_bsr_border_set

This function configures a BSR (bootstrap router) border on the VIF.

Syntax

```
int
pim4_api_vif_bsr_border_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_bsr_border_unset

This function removes configuration of a BSR border from the VIF.

Syntax

```
int
pim4_api_vif_bsr_border_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_unbind_bundle

This function unbinds the interface from an ECMP Bundle.

Syntax

```
int
pim4_api_vif_unbind_bundle (u_int32_t vr_id, char *ifname, char *bundle_name)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`ifname`	Interface Name
`bundle_name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when PIM VIF has not been created for the interface

PIM_API_SET_ERR_NO_MATCH_FOR_CONFIGURED_VALUE when the bundle name does not match the configured value

PIM_API_SET_ERR_COMMAND_NOT_CONFIGURED The interface has not been bound to any bundle.

PIM_API_SUCCESS when the function executes properly

pim4_api_vif_dr_priority_set

This function configures the priority for a PIM Designated Router (DR) on a VIF.

Syntax

```
int
pim4_api_vif_dr_priority_set (u_int32_t vr_id, char *ifname,
                              u_int32_t priority);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`priority`	The priority of the Designated Router

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_dr_priority_unset

This function removes the PIM DR priority from the VIF and returns it to the default setting.

Syntax

```
int
pim4_api_vif_dr_priority_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_exclude_genid_set

This function configures PIM to exclude a generated ID on the VIF.

Syntax

```
int
pim4_api_vif_exclude_genid_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_exclude_genid_unset

This function removes the configuration to exclude a generated ID on the VIF.

Syntax

```
int
pim4_api_vif_exclude_genid_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_hello_interval_set

This function configures a PIM hello interval on a VIF.

Syntax

```
int
pim4_api_vif_hello_interval_set (u_int32_t vr_id, char *ifname,
                                 u_int16_t val);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of interface
`val`	Hello interval in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_hello_interval_unset

This function removes the configuration for a PIM hello interval on a VIF and resets the hello interval to its default value.

Syntax

```
int
pim4_api_vif_hello_interval_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_hello_holdtime_set

This function configures a PIM hello holdtime, in seconds, for a VIF.

Syntax

```
int
pim4_api_vif_hello_holdtime_set (u_int32_t vr_id, char *ifname,
                                 u_int16_t val);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`val`	Hello-holdtime interval in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_hello_holdtime_unset

This function removes the configuration of the PIM hello holdtime from a VIF and resets it to its default value.

Syntax

```
int
pim4_api_vif_hello_holdtime_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_mode_set

This function configures the PIM mode on a virtual interface (VIF).

Syntax

```
int
pim4_api_vif_mode_set (u_int32_t vr_id, char *ifname,
                       pim_api_mode_t mode);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`mode`	PIM mode set DM or SM
`DM`	Dense mode
`SM`	Sparse mode
`SMDM`	Sparse-dense mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim4_api_vif_mode_unset

This function removes the PIM mode configuration from a VIF.

Syntax

```
int
pim4_api_vif_mode_unset (u_int32_t vr_id, char *ifname, pim_api_mode_t mode)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`mode`	PIM mode set, either
`DM`	Dense mode
`SM`	Sparse mode
`SMDM`	Sparse-dense mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_ERR_MODE_MIS_MATCH when there is mismatch while comparing the configured mode and user command

PIM_API_SUCCESS when the function executes properly

pim4_api_vif_nbr_filter_set

This function configures a PIM neighbor filter access-list name for a VIF.

Syntax

```
int
pim4_api_vif_nbr_filter_set (u_int32_t vr_id, char *ifname,
                             char *filter);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_nbr_filter_unset

This function removes the configuration for a PIM VIF neighbor filter access-list name.

Syntax

```
int
pim4_api_vif_nbr_filter_unset (u_int32_t vr_id, char *ifname, char *filter)
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_passive_set

This function configures a PIM virtual interface as passive.

Syntax

```
int
pim4_api_vif_passive_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_passive_unset

This function removes the configuration for a passive PIM VIF.

Syntax

```
int
pim4_api_vif_passive_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_propagation_delay_set

This function configures the PIM propagation delay, in milliseconds, for a VIF.

Syntax

```
int
pim4_api_vif_propagation_delay_set (u_int32_t vr_id, char *ifname,
                                    u_int16_t val);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`val`	Propagation delay in milliseconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_propagation_delay_unset

This function removes the PIM propagation delay from the VIF, and returns the value to its default setting.

Syntax

```
int
pim4_api_vif_propagation_delay_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_state_refresh_originate_interval_set

This function configures the state refresh that originates the interval for PIM-DM.

Syntax

```
int
pim4_api_vif_state_refresh_originate_interval_set (u_int32_t vr_id,
                                                   char *ifname,
                                                   u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`sec`	State refresh original interval in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_state_refresh_originate_interval_unset

This function removes the configuration of the state-refresh interval for PIM-DM.

Syntax

```
int
pim4_api_vif_state_refresh_originate_interval_unset (u_int32_t vr_id,
                                                     char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_unicast_bsm_set

This AP configures PIM to send a unicast BSM (broadband satellite multimedia) message to the VIF.

Syntax

```
int
pim4_api_vif_unicast_bsm_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim4_api_vif_unicast_bsm_unset

This function removes the PIM configuration to send unicast BSM messages to the VIF.

Syntax

```
int
pim4_api_vif_unicast_bsm_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

CHAPTER 5 PIM6 Command API

The functions in this chapter are called by the PIM IPv6 commands.

pim6_api_anycast_rp_set

This function configures a set of anycast RPs.

Syntax

```
int
pim6_api_anycast_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                         struct pal_in6_addr *anycast_rp_addr,
                         struct pal_in6_addr *member_rp_addr);
```

Input Parameters

`vr_id` Virtual router ID

`vrf_id` VPN routing/forwarding instance ID

Output Parameters

`anycast_rp_addr`

Address of the anycast RP

`member_rp_addr`

Communication IP address between the configured RPs in the RP set

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_WRONG_VALUE when anycast address is not found

PIM_API_SET_ERR_RP_ANYCAST_SHOULD_BE_UNICAST when the anycast address in not unicast address

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_anycast_rp_unset

This function removes the configuration of an anycast RP set.

Syntax

```
int
pim6_api_anycast_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                           struct pal_in6_addr *anycast_rp_addr,
                           struct pal_in6_addr *member_rp_addr);
```

Input Parameters

`vr_id` Virtual router ID

`vrf_id` VPN routing/forwarding instance ID

Output Parameters

`anycast_rp_addr`

Address of the anycast RP

`member_rp_addr`

Communication IP address between the configured RPs in the RP set

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_WRONG_VALUE when anycast address is not found

PIM_API_SET_ERR_RP_ANYCAST_SHOULD_BE_UNICAST when the anycast address in not unicast address

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bidir_disable

This function disables bidirectional PIM.

Syntax

```
int
pim6_api_bidir_disable (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id` Virtual router ID(0-255)

`vrf_id` VPN Routing/Forwarding Instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_bidir_enable

This function enables bidirectional PIM.

Syntax

```
int
```



```
pim6_api_bidir_enable (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

vr_id	Virtual router ID(0-255)
vrf_id	VPN Routing/Forwarding Instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_bsr_candidate_set

This function sets the specified router as candidate BSR using the interface name as its address.

Syntax

```
int
pim6_api_bsr_candidate_set (u_int32_t vr_id, vrf_id_t vrf_id,
                            char *ifname);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	The name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_candidate_unset

This function removes the specified router as candidate BSR.

Syntax

```
int
pim6_api_bsr_candidate_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                              char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	The name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_candidate_hash_mask_set

This function configures the hash-mask length for a candidate BSR.

Syntax

```
int
pim6_api_bsr_candidate_hash_mask_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      char *ifname, u_char hash_mask);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`hash_mask`	Hash mask length used to hash for RPs

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_candidate_hash_mask_unset

This function removes configuration of a hash-mask length for a candidate BSR.

Syntax

```
int
pim6_api_bsr_candidate_hash_mask_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_candidate_priority_set

This function configures the priority of a candidate BSR.

Syntax

```
int
pim6_api_bsr_candidate_priority_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     char *ifname, u_char priority);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`priority`	Priority value assigned to the BSR

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_candidate_priority_unset

This function removes the configuration of a priority value for a candidate BSR.

Syntax

```
int
pim6_api_bsr_candidate_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_interop_set

This function configures bootstrap router interoperability.

Syntax

```
int
pim6_api_bsr_interop_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_bsr_interop_unset

This function removes configuration of BSR interoperability.

Syntax

```
int
pim6_api_bsr_interop_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_clear_bsr_rpset

This function clears the specified router as the candidate BSR RP set.

Syntax

```
int
pim6_api_clear_bsr_rpset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_clear_tib

This function clears all TIB states and MFC cache entries.

Syntax

```
int
pim6_api_clear_tib (u_int32_t vr_id, vrf_id_t vrf_id, pim_api_mode_t mode,
                    struct pal_in6_addr *src, struct pal_in6_addr *grp);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`mode`	Mode of operation, either dense more or sparse mode

Output Parameters

`*src`	Source address of the TIB to clear

`*grp`	Group address of the TIB to clear

Return Value

IM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_crp_per_grp_chk

Syntax

```
int
pim6_api_crp_per_grp_chk (u_int32_t vr_id, vrf_id_t vrf_id,
                          char *ifname, char *group_acl_name)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`group_acl_name`	Name of the ACL to use a the group range

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

PIM_API_CHECK_ERR_CRP_IF_OR_GROUP_IN_USED

pim6_api_debug_all_set

This function enables debugging for all PIM events.

Syntax

```
int
pim6_api_debug_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_all_unset

This function disables debugging for all PIM events.

Syntax

```
int
pim6_api_debug_all_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_event_set

This function enables debugging of a PIM event.

Syntax

```
int
pim6_api_debug_event_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_event_unset

This function disables debugging of a PIM event.

Syntax

```
int
pim6_api_debug_event_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_nsm_set

This function enables debugging of NSM events.

Syntax

```
int
pim6_api_debug_nsm_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_nsm_unset

This function disables debugging of NSM events.

Syntax

```
int
pim6_api_debug_nsm_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_all_set

This function enables debugging of all PIM packets.

Syntax

```
int
pim6_api_debug_packet_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_all_unset

This function disables debugging of all PIM packets.

Syntax

```
int
pim6_api_debug_packet_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_in_set

This function enables debugging of incoming PIM packets.

Syntax

```
int
pim6_api_debug_packet_in_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_in_unset

This function disables debugging of an incoming PIM packet.

Syntax

```
int
pim6_api_debug_packet_in_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_out_set

This function enables debugging of an outgoing PIM packet.

Syntax

```
int
pim6_api_debug_packet_out_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_packet_out_unset

This function disables debugging of an outgoing PIM packet.

Syntax

```
int
pim6_api_debug_packet_out_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_nexthop_set

This function enables debugging of Reverse Path Forwarding (RPF) nexthop cache handling.

Syntax

```
int
pim6_api_debug_nexthop_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_nexthop_unset

This function disables debugging of RPF nexthop cache handling.

Syntax

```
int
pim6_api_debug_nexthop_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mfc_set

This function enables debugging of MFC updates.

Syntax

```
int
pim6_api_debug_mfc_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mfc_unset

This function disables debugging of MFC updates.

Syntax

```
int
pim6_api_debug_mfc_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mib_set

This function enables debugging of MIB entries.

Syntax

```
int
pim6_api_debug_mib_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mib_unset

This function disables debugging of MIB entries.

Syntax

```
int
pim6_api_debug_mib_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mtrace_set

This function enables debugging of MTRACE messages.

Syntax

```
int
pim6_api_debug_mtrace_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_mtrace_unset

This function disables debugging of MTRACE messages.

Syntax

```
int
pim6_api_debug_mtrace_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_state_set

This function enables debugging of the PIM state.

Syntax

```
int
pim6_api_debug_state_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_state_unset

This function disables debugging of the PIM state.

Syntax

```
int
pim6_api_debug_state_unset (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_all_set

This function enables debugging of all timers.

Syntax

```
int
pim6_api_debug_timer_all_set (u_int32_t vr_id, vrf_id_t vrf_id, int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_all_unset

This function disables debugging of all timers.

Syntax

```
int
pim6_api_debug_timer_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_assert_all_set

This function enables debugging of all Assert timers.

Syntax

```
int
pim6_api_debug_timer_assert_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_assert_all_unset

This function disables debugging of all Assert timers.

Syntax

```
int
pim6_api_debug_timer_assert_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_assert_timer_set

This function enables debugging of the PIM assert timer.

Syntax

```
int
pim6_api_debug_timer_assert_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                       int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_assert_timer_unset

This function disables debugging of the PIM assert timer.

Syntax

```
int
pim6_api_debug_timer_assert_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                         int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_all_set

This function enables debugging of the Bootstrap Router times.

Syntax

```
int
pim6_api_debug_timer_bsr_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                  int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_all_unset

This function disables debugging of the BSR timers.

Syntax

```
int
pim6_api_debug_timer_bsr_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_bootstrap_set

This function enables debugging of the BSR timer.

Syntax

```
int
pim6_api_debug_timer_bsr_bootstrap_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_bootstrap_unset

This function disables debugging of the BSR timer.

Syntax

```
int
pim6_api_debug_timer_bsr_bootstrap_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_candidate_rp_set

This function enables debugging of the BSR Candidate-RP timer.

Syntax

```
    int
pim6_api_debug_timer_bsr_candidate_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                           int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_bsr_bootstrap_unset

This function disables debugging of the BSR Candidate-RP timer.

Syntax

```
int
pim6_api_debug_timer_bsr_candidate_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                             int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_all_set

This function enables debugging of the hello timers.

Syntax

```
int
pim6_api_debug_timer_hello_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_all_unset

This function disables debugging of the hello timers.

Syntax

```
int
pim6_api_debug_timer_hello_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_timer_set

Syntax

```
int
pim6_api_debug_timer_hello_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_timer_unset

Syntax

```
int
pim6_api_debug_timer_hello_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_neighbor_liveliness_set

This function enables debugging of the neighbor-liveliness hello timer.

Syntax

```
int
pim6_api_debug_timer_hello_neighbor_liveliness_set (u_int32_t vr_id,
                                                    vrf_id_t vrf_id,
                                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_neighbor_liveliness_unset

This function disables debugging of the neighbor-liveliness hello timer.

Syntax

```
int
pim6_api_debug_timer_hello_neighbor_liveliness_unset (u_int32_t vr_id,
                                                      vrf_id_t vrf_id,
                                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_triggered_set

This function enables debugging of the triggered-hello timer.

Syntax

```
int
pim6_api_debug_timer_hello_triggered_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_hello_triggered_unset

This function disables debugging of the triggered-hello timer.

Syntax

```
int
pim6_api_debug_timer_hello_triggered_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                            int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_all_set

This function enables debugging of all join/prune timers.

Syntax

```
int
pim6_api_debug_timer_jp_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                 int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_all_unset

This function disables debugging of all join/prune timers.

Syntax

```
int
pim6_api_debug_timer_jp_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                   int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_timer_set

This function enables debugging of the join/prune timer.

Syntax

```
int
pim6_api_debug_timer_jp_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                   int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_timer_unset

This function disables debugging of the join/prune timer.

Syntax

```
int
pim6_api_debug_timer_jp_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                     int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_expiry_set

This function enables debugging of the join/prune expiration timer.

Syntax

```
int
pim6_api_debug_timer_jp_expiry_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_expiry_unset

This function disables debugging of the join/prune expiration timer.

Syntax

```
int
pim6_api_debug_timer_jp_expiry_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_prune_pending_set

This function enables debugging of the join/prune pending timer.

Syntax

```
int
pim6_api_debug_timer_jp_prune_pending_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                           int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_prune_pending_unset

This function disables debugging of the join/prune pending timer.

Syntax

```
int
pim6_api_debug_timer_jp_prune_pending_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                             int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_keep_alive_set

This function enables debugging of the join/prune keepalive timer.

Syntax

```
int
pim6_api_debug_timer_jp_keep_alive_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_keep_alive_unset

This function disables debugging of the join/prune keepalive timer.

Syntax

```
int
pim6_api_debug_timer_jp_keep_alive_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_override_set

This function enables debugging of the join/prune upstream override timer.

Syntax

```
int
pim6_api_debug_timer_jp_override_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_jp_override_unset

This function disables debugging of the join/prune upstream override timer.

Syntax

```
int
pim6_api_debug_timer_jp_override_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
cli_mode	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_register_all_set

This function enables debugging of the Register timers.

Syntax

```
int
pim6_api_debug_timer_register_all_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                       int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_register_all_unset

This function disables debugging of the Register timers.

Syntax

```
int
pim6_api_debug_timer_register_all_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                         int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_register_stop_set

This function enables debugging of the Register Stop timer.

Syntax

```
int
pim6_api_debug_timer_register_stop_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_debug_timer_register_stop_unset

This function disables debugging of the Register Stop timer.

Syntax

```
int
pim6_api_debug_timer_register_stop_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          int cli_mode);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`cli_mode`	CLI command mode

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_df_offer_limit_set

This function configures the limit for a Designated Forwarder (DF) Offer Message

Syntax

```
int
pim6_api_df_offer_limit_set (u_int32_t vr_id, vrf_id_t vrf_id,
                             u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer limit

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_df_offer_interval_set

This function configures the interval for a Designated Forwarder (DF) Offer Message.

Syntax

```
int
pim6_api_df_offer_interval_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                u_int32_t val);
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`val`	Value of the DF offer interval

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_df_offer_interval_unset

This function unconfigures the interval for a Designated Forwarder (DF) Offer Message.

Syntax

```
int
pim6_api_df_offer_interval_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                  u_int32_t val);
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
val	Value of the DF offer interval

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_dm_group_default_set

This function sets/unsets a default dense-group address

Syntax

```
int
pim6_api_dm_group_default_set (u_int32_t vr_id, vrf_id_t vrf_id,
                               struct pal_in4_addr *grp_addr, u_int32_t set)
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
grp_addr	IPv6 address of dense-group
set	Add/Delete the dense-group

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_DM_MEMBER_ALREADY_PRESENT when the group address has already been configured

PIM_API_SUCCESS when the function executes properly

pim6_api_ecmp_bundle_create

This function creates an ECMP bundle.

Syntax

```
int
pim6_api_ecmp_bundle_create (u_int32_t vr_id, vrf_id_t vrf_id, char *name);
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
name	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM VR is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_NAME_TOO_LONG when the bundle name exceeds the limit of 50 characters.

PIM_API_SET_ERR_OUT_OF_MEMORY when the process has run out of memory

PIM_API_SET_ERR_ECMP_BUNDLE_EXISTS when the bundle already exists

PIM_API_SUCCESS when the function executes properly

pim6_api_ecmp_bundle_delete

This function deletes an ECMP bundle.

Syntax

```
int
pim6_api_ecmp_bundle_delete (u_int32_t vr_id, vrf_id_t vrf_id, char *name);
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
name	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ECMP_BUNDLE_NOT_FOUND when the specified bundle cannot be found

PIM_API_SUCCESS when the function executes proper

pim6_api_embed_rp_unset

Syntax

```
int
pim6_api_embed_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ignore_rp_set_priority_unset

This function removes the configuration to ignore the RP priority while performing an RP selection.

Syntax

```
int
pim6_api_ignore_rp_set_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_join_prune_timer_set

This function configures the PIM6 join/prune timer.

Syntax

```
int
pim6_api_join_prune_timer_set (u_int32_t vr_id, vrf_id_t vrf_id,
                               u_int32_t timer);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`timer`	Join/prune timer value, in seconds

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_join_prune_timer_unset

This function removes configuration of the PIM6 join/prune timer.

Syntax

```
int
pim6_api_join_prune_timer_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ignore_rp_set_priority_set

This function configures PIM6 to ignore the RP priority while performing an RP selection.

Syntax

```
int
pim6_api_ignore_rp_set_priority_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_spt_switch_threshold_set

This function configures an SPT (system posture token) switchover threshold. The optional group list ACL filters the groups for which SPT switchover is performed.

Syntax

```
int
pim6_api_spt_switch_threshold_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                   char *group_list_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`group_list_acl_name`	Name of the access control list to use as the group range

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_source_address_set

This function configures the source address of Register messages.

Syntax

```
int
pim6_api_register_source_address_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                      struct pal_in6_addr *source_addr);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`source_addr`	IP address to use as the source address for Registers

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_source_interface_set

This function sets the source interface of register messages to the address of the given interface.

Syntax

```
int
pim6_api_register_source_interface_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Interface name to set.

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_source_unset

This function removes the configuration of a source address for register messages. After this call, the source address of register messages is the address of the interface over which the data packet that generates the register message has arrived.

Syntax

```
int
pim6_api_register_source_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_rate_limit_set

This function configures the maximum number of Registers to generate for an (S,G).

Syntax

```
int
pim6_api_register_rate_limit_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                  u_int16_t rate_limit);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`rate_limit`	Maximum number of Register messages allowed, in packets per second

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_rate_limit_unset

This function removes the configuration of a maximum number of Register messages allowed for an (S,G).

Syntax

```
int
pim6_api_register_rate_limit_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_rp_reachability_check_set

This function configures an RP unicast reachability check in the Register state machine.

Syntax

```
int
pim6_api_register_rp_reachability_check_set (u_int32_t vr_id,
                                             vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_rp_reachability_check_unset

This function removes configuration of an RP unicast reachability check in the Register state machine.

Syntax

```
int
pim6_api_register_rp_reachability_check_unset (u_int32_t vr_id,
                                               vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_set_all

This function configures an RP candidate.

Syntax

```
int
pim6_api_rp_candidate_set_all (u_int32_t vr_id, vrf_id_t vrf_id,
                               char *ifname, bool_t is_bidir, char *acl,
                               s_int32_t interval,s_int32_t priority);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Interface name
`is_bidir`	Whether bidirectional
`acl`	Access-list name
`interval`	Interval value of RP candidate
`priority`	Priority value of RP candidate

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_register_keep_alive_timer_set

This function configures the keepalive timer (KAT) value of (S,G) created by the RP by Register messages.

Syntax

```
int
pim6_api_rp_register_keep_alive_timer_set (u_int32_t vr_id,
                                           vrf_id_t vrf_id,
                                           u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`sec`	Keepalive-timer value, in seconds

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_register_keep_alive_timer_unset

This function removes configuration of the KAT value of (S,G) created at the RP by Register messages. The KAT value is then reset to (Register Suppression Time * 3) + Register Probe Interval.

Syntax

```
int
pim6_api_rp_register_keep_alive_timer_unset (u_int32_t vr_id,
                                             vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_suppression_time_set

This function configures the Register Suppression interval.

Syntax

```
int
pim6_api_register_suppression_time_set (u_int32_t vr_id,
                                        vrf_id_t vrf_id,
                                        u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`sec`	Register suppression interval, in seconds

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_register_suppression_time_unset

This function removes configuration of a register suppression interval

Syntax

```
int
pim6_api_register_suppression_time_unset (u_int32_t vr_id,
                                          vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance DI

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_accept_register_filter_set

This function configures an ACL to filter sources allowed to register with this RP.

Syntax

```
int
pim6_api_rp_accept_register_filter_set (u_int32_t vr_id,
                                        vrf_id_t vrf_id,
                                        char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_accept_register_filter_unset

This function removes the configuration of an ACL to filter sources allowed to register with this RP.

Syntax

```
int
pim6_api_rp_accept_register_filter_unset (u_int32_t vr_id,
                                          vrf_id_t vrf_id,
                                          char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_set

This function configures an RP candidate set.

Syntax

```
int
pim6_api_rp_candidate_set (u_int32_t vr_id, vrf_id_t vrf_id,
                           char *ifname);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_unset

This function removes configuration of a candidate RP set.

Syntax

```
int
pim6_api_rp_candidate_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                             char *ifname);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
ifname	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_adv_interval_set

This function configures the advertisement interval for a candidate RP.

Syntax

```
int
pim6_api_rp_candidate_adv_interval_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                        char *ifname, u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`sec`	The C-RP advertisement interval, in seconds

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_adv_interval_unset

This function removes a candidate RP advertisement interval.

Syntax

```
int
pim6_api_rp_candidate_adv_interval_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                          char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_priority_set

This function configures an RP candidate set and sets its priority,

Syntax

```
int
pim6_api_rp_candidate_priority_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                    char *ifname, u_char priority);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`priority`	Priority value of the RP candidate set

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_priority_unset

This function removes the configuration for a candidate RP set and its priority.

Syntax

```
int
pim6_api_rp_candidate_priority_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                      char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_group_acl_set

This function configures a group range using an access control list for a candidate RP set.

Syntax

```
int
pim6_api_rp_candidate_group_acl_set (u_int32_t vr_id, vrf_id_t vrf_id,
                                     char *ifname, char *group_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface
`group_acl_name`	Name of the ACL to use a the group range

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_candidate_group_acl_unset

This function removes the configuration of a group range for a candidate RP set.

Syntax

```
int
pim6_api_rp_candidate_group_acl_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                                       char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`ifname`	Name of the interface

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_checksum_filter_set

This function configures a Cisco-style checksum. The filter ACL controls the groups to which the checksum calculation is applied.

Syntax

```
int
pim6_api_rp_checksum_filter_set (u_int32_t vr_id,
                                 vrf_id_t vrf_id,
                                 char *acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`acl_name`	Access control list name

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_rp_checksum_filter_unset

This function removes the configuration of a Cisco-style Register checksum.

Syntax

```
int
pim6_api_rp_checksum_filter_unset (u_int32_t vr_id,vrf_id_t vrf_id,char *acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`acl_name`	Access control list name

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_router_id_set

This function sets a router-id.

Syntax

```
int
pim6_api_router_id_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        struct pal_in4_addr router_id)
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
router_id	Router-ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM VR is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ROUTER_ID_INVALID when the router-id cannot be configured with the value

PIM_API_SUCCESS when the function executes properly

pim6_api_router_id_unset

This function deletes the router-id.

Syntax

```
int
pim6_api_router_id_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          struct pal_in6_addr router_id)
```

Input Parameters

vr_id	Virtual router ID (0-255)
vrf_id	VPN Routing/Forwarding Instance ID
router_id	Router-ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when the interface name does not exists

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_ROUTER_ID_INVALID when the router-id cannot be configured with the value

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ssm_default_set

This function configures PIM SSM operation.

Syntax

```
int
pim6_api_ssm_default_set (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ssm_default_unset

This function removes the configuration for PIM SSM operation.

Syntax

```
int
pim6_api_ssm_default_unset (u_int32_t vr_id, vrf_id_t vrf_id);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ssm_range_set

This function configures an SSM range using an access control list.

Syntax

```
int
```

```
pim6_api_ssm_range_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	Access control list name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_ssm_range_unset

This function removes configuration of an SSM range using an ACL.

Syntax

```
int
pim6_api_ssm_range_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          char *acl_name);
```

Input Parameters

vr_id	Virtual router ID
vrf_id	VPN routing/forwarding instance ID
acl_name	ACL name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_static_rp_set

This function configures a static rendezvous point (RP).

Syntax

```
int
pim6_api_static_rp_set (u_int32_t vr_id, vrf_id_t vrf_id,
                        struct pal_in6_addr *rp_addr,
```

```
                        char *acl_name, bool_t override_flag);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing//forwarding instance ID
`acl_name`	Name of ACL (access control list) to use.
`override_flag`	Override flag for the interface

Output Parameters

`rp_addr`	Rendezvous point address

Return Value

PIM_API_SET_ERR_WRONG_VR

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_static_rp_unset

This function removes a static RP configuration.

Syntax

```
int
pim6_api_static_rp_unset (u_int32_t vr_id, vrf_id_t vrf_id,
                          struct pal_in6_addr *rp_addr);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

`rp_addr`	Address of the RP

Return Value

PIM_API_SET_ERR_WRONG_VR

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_spt_switch_threshold_unset

This function removes the configuration of an SPT switchover threshold.

Syntax

```
int
pim6_api_spt_switch_threshold_unset (u_int32_t vr_id, vrf_id_t vrf_id,
```

```
                                    char *group_list_acl_name);
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID
`group_list_acl_name`	Name of the access control list used as the group range

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR when PIM Master is not found

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_vif_bidir_nbr_filter_set

This function configures the PIM VIF bidirectional neighbor filter access-list name

Syntax

```
int
pim6_api_vif_bidir_nbr_filter_set (u_int32_t vr_id, char *ifname,
                                   char *filter);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_vif_bidir_nbr_filter_unset

This function unconfigures the PIM VIF bidirectional neighbor filter access-list name

Syntax

```
int
pim6_api_vif_bidir_nbr_filter_unset (u_int32_t vr_id, char *ifname,
                                     char *filter);
```

Input Parameters

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

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERROR when BSR information is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_vif_bind_bundle

This function binds the interface to an ECMP Bundle.

Syntax

```
int
pim6_api_vif_bind_bundle (u_int32_t vr_id, char *ifname, char *bundle_name)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`ifname`	Interface Name
`bundle_name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SUCCESS when the function executes properly

pim6_api_vif_unbind_bundle

This function unbinds the interface to an ECMP Bundle.

Syntax

```
int
pim6_api_vif_unbind_bundle (u_int32_t vr_id, char *ifname, char *bundle_name)
```

Input Parameters

`vr_id`	Virtual router ID (0-255)
`vrf_id`	VPN Routing/Forwarding Instance ID
`ifname`	Interface Name
`bundle_name`	ECMP Bundle name

Output Parameters

None

Return Values

PIM_API_SET_ERR_WRONG_VR when the interface name does not exists

PIM_API_SET_ERR_VIF_NOT_EXIST when the PIM VIF has not been created for the interface

PIM_API_SET_ERR_COMMAND_NOT_CONFIGURED The interface has not been bound to any bundle.

PIM_API_SET_ERR_NO_MATCH_FOR_CONFIGURED_VALUE when the bundle name does not match the configured value

PIM_API_SUCCESS when the function executes properly

pim6_api_vif_passive_set

This function configures a PIM VIF interface as passive.

Syntax

```
int
pim6_api_vif_passive_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_passive_unset

This function removes the configuration of a PIM VIF aa a passive interface.

Syntax

```
int
pim6_api_vif_passive_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_hello_interval_set

This function configures the PIM VIF hello interval seconds.

Syntax

```
int
pim6_api_vif_hello_interval_set (u_int32_t vr_id, char *ifname,
                                 u_int16_t val);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`val`	Hello interval in seconds.

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_hello_interval_unset

This function unconfigures the PIM VIF hello interval and make it default.

Syntax

```
int
```

```
pim6_api_vif_hello_interval_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_hello_holdtime_set

This function configures the PIM VIF hello holdtime seconds.

Syntax

```
int
pim6_api_vif_hello_holdtime_set (u_int32_t vr_id, char *ifname,
                                 u_int16_t val)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`val`	Hello interval in seconds.

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_hello_holdtime_unset

This function removes the configuration for the PIM VIF hello holdtime and returns it to its default setting.

Syntax

```
int
pim6_api_vif_hello_holdtime_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_propagation_delay_set

This function configures the PIM VIF propagation delay in milliseconds.

Syntax

```
int
pim6_api_vif_propagation_delay_set (u_int32_t vr_id, char *ifname,
                                    u_int16_t val);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`val`	Propagation delay in milliseconds

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_propagation_delay_unset

This function removes configuration of the PIM VIF propagation delay and returns the value to its default setting.

Syntax

```
int
pim6_api_vif_propagation_delay_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when PIM VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_mode_set

This function configures the PIM mode on VIF interface

Syntax

```
int
pim6_api_vif_mode_set (u_int32_t vr_id, char *ifname,
                       pim_api_mode_t mode);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`mode`	PIM mode, either dense-mode or sparse-mode

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_mode_unset

This function removes the configured PIM mode from a VIF interface.

Syntax

```
int
pim6_api_vif_mode_unset (u_int32_t vr_id, char *ifname, pim_api_mode_t mode)
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`mode`	PIM mode, either dense-mode or sparse-mode or sparse-dense mode

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

PIM_API_SET_ERR_MODE_MIS_MATCH when there is mismatch while comparing the configured mode and user command

pim6_api_vif_nbr_filter_set

This function configures a PIM VIF neighbor filter access control list on the interface.

Syntax

```
int
pim6_api_vif_nbr_filter_set (u_int32_t vr_id, char *ifname,
                             char *filter);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`filter`	Name of ACL to use as neighbor filter

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_nbr_filter_unset

This function removes the configuration of a PIM VIF neighbor filter access control list name.

Syntax

```
int
pim6_api_vif_nbr_filter_unset (u_int32_t vr_id, char *ifname,
                               char *filter);
```

Input Parameters

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

`filter`	Name of the ACL used as a neighbor filter

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly.

pim6_api_vif_state_refresh_originate_interval_set

This function configures the state refresh that originates the interval for PIM-DM.

Syntax

```
int
pim6_api_vif_state_refresh_originate_interval_set (u_int32_t vr_id,
                                                   char *ifname,
                                                   u_int16_t sec);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`sec`	State refresh interval in seconds.

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_vif_state_refresh_originate_interval_unset

This function removes configuration of the state refresh originate interval for PIM-DM.

Syntax

```
int
pim6_api_vif_state_refresh_originate_interval_unset (u_int32_t vr_id,
                                                     char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface
`sec`	State refresh interval in seconds.

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly

pim6_api_vif_dr_priority_set

This function configures the PIM VIF DR priority for the interface. When the interface is enabled, DR selection is also performed.

Syntax

```
int
pim6_api_vif_dr_priority_set (u_int32_t vr_id, char *ifname,
                              u_int32_t priority);
```

Input Parameters

`vr_id`	Virtual router ID
`ifname`	Name of the interface
`priority`	DR priority value

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_dr_priority_unset

This function remove the configuration for PIM VIF DR priority and returns it to its default setting.

Syntax

```
int
pim6_api_vif_dr_priority_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	Virtual router ID

`ifname`	Name of the interface

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_exclude_genid_set

This function configures PIM to exclude a generated ID on the VIF.

Syntax

```
int
pim6_api_vif_exclude_genid_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_exclude_genid_unset

This function remove the setting to exclude a generated ID on the VIF.

Syntax

```
int
pim6_api_vif_exclude_genid_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_bsr_border_set

This function configure a BSR border on the VIF.

Syntax

```
int
pim6_api_vif_bsr_border_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_bsr_border_unset

This function removes the configuration for a BSR border from the VIF.

Syntax

```
int
pim6_api_vif_bsr_border_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_unicast_bsm_set

This function configures PIM to send a unicast BSM (broadband satellite multimedia) message to the VIF.

Syntax

```
int
pim6_api_vif_unicast_bsm_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly when the function executes properly

pim6_api_vif_unicast_bsm_unset

This function removes the configuration for PIM to send unicast BSM messages to the VIF.

Syntax

```
int
pim6_api_vif_unicast_bsm_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

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

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VALUE when an invalid value is given

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_ERR_VIF_NOT_EXIST when VIF is not found

PIM_API_SET_SUCCESS when the function executes properly.

pim6_api_embed_rp_set

Syntax

```
int
pim6_api_embed_rp_set (u_int32_t vr_id, vrf_id_t vrf_id)
```

Input Parameters

`vr_id`	Virtual router ID
`vrf_id`	VPN routing/forwarding instance ID

Output Parameters

None

Return Value

PIM_API_SET_ERR_WRONG_VR

PIM_API_SET_ERR_WRONG_VRF when PIM VRF interface is not found

PIM_API_SET_SUCCESS when the function executes properly.

Index

R

V