ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Routing Information Protocol
Developer Guide

December 2015

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

Contents

Preface

This guide describes the ZebOS-XP application programming interface (API) for Router Information Protocol (RIP).

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This document contains these chapters and appendices:

- Chapter 1, *Introduction*
- Chapter 2, *Data Structures*
- Chapter 3, *RIP Command API*
- Chapter 4, *RIPng Command API*
- Chapter 5, *SNMP API*
- Appendix A, *Source Files*

Related Documents

The following guides are related to this document:

- *Routing Information Protocol Command Reference*
- *Network Services Module Command Reference*
- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 Introduction

This chapter provides an overview of the Routing Information Protocol (RIP).

Overview

The RIP and RIPng protocol modules are portable software that implement the industry-standard Routing Information Protocol. The RIP and RIPng modules are built on the Network Services Module (NSM) and are IETF-compliant. An extensive set of features are supported. This control-plane software module can integrate into a range of network processor environments.

RIP Features

Features Summary

- IPv4
- Static Routes Support
- Implements RIPv1 and RIPv2 Protocols
- RIP Neighbor Configuration Support
- Simple Password Authentication
- Basic Timer Configuration
- MD5 Authentication Support
- Offset-List Support
- Variable Length Subnet Masks Support
- Split Horizon Support
- Poison Reverse Support
- SNMP API
- Industry Standard Command Line Interface
- RIPv2-MIB With RIP Peer Support
- Restart Support
- RIP CE-PE Support
- Virtual Routing Support
- Flag to Control Metric Updates

RFC Standards Support

- RFC 1058 — Routing Information Protocol (RIP)
- RFC 1724 — RIP Management Information Base (MIB)
- RFC 2082 — RIP-2 MD5 Authentication
- RFC 2453 — Routing Information Protocol Version 2 (RIPv2)

RIPng Features

Features Summary

- IPv6 Support
- Static Routes Support
- Basic Timer Configuration Support
- Route Filtering
- Route Redistribution
- RIP Neighbor Configuration Support
- Split Horizon Support
- Poison Reverse Support
- Offset List Support
- Industry-standard command line interface (CLI)
- Flag to Control Metric Updates

RFC Standards Support

- RFC 2080 — Routing Information Protocol Next Generation (RIPng)
- RFC 2453 — Routing Information Protocol Version 2 (RIPv2)

Architecture

This section describes the RIP and RIPng interfaces with other ZebOS-XP components such as NSM, IMI, and the TCP/IP stack.

Architecture Overview

The RIP/RIPng module has the following interfaces to external components in ZebOS-XP:

- NSM Inter Process Communication
- IIMI Inter Process Communication
- Platform Abstraction Layer Interface

For information about messages, refer to the *Network Services Module Developer Guide.*

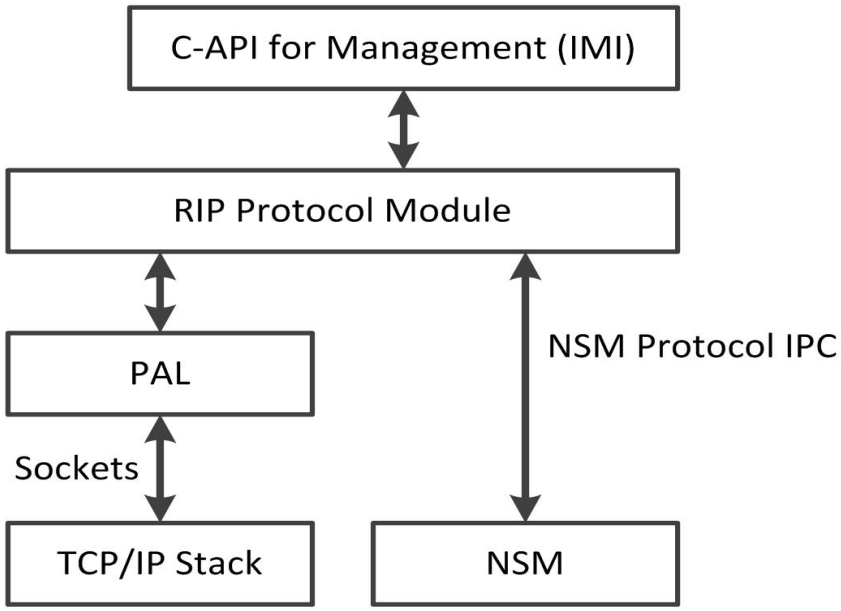

Figure 1-1: RIP Architecture

NSM Inter Process Communication

RIP extends an NSM client interface to handle Inter Process Communication between NSM and RIP. NSM shares the following information with RIP:

- Layer 3 Interface Information
- Redistributed Routes (Static, OSPF)

IMI Inter Process Communication

RIP extends an IMI client interface to handle Inter Process Communication between IMI and RIP. The IPC between IMI and RIP is established to exchange the following information:

- CLI configuration from IMI to the RIP module.
- Result of CLI configuration from the RIP to IMI module

Platform Abstraction Layer Interface

The Platform Abstraction Layer (PAL) interface in RIP is the interface to the TCP/IP stack to send and receive control packets.

Metric Behavior Types

Two metric updating behavior types are supported:

- Per standard behavior: RFC 2453 or RFC 2080 (default)
- Per Cisco behavior

By default, Cisco metric-behavior is disabled. Cisco metric-behavior is enabled using the `enable` command. The default metric-behavior type is set using the `disable` or `no` command.

Metric behavior types are controlled by a metric-update flag: for standard behavior, the flag is set to RIP_RFC_METRIC_COMPLIANT; for Cisco behavior, the flag is set to RIP_CISCO_METRIC_COMPLIANT.

Standard Metric Behavior

When the RIP process receives a response message, the metric-update flag is checked: if set to RIP_RFC_METRIC COMPLIANT, it increments the metric by adding the cost of the network on which the message arrived. It then adds the entry to the database, or updates the entry in the database if the entry already exists.

When sending a response message, the metric-update flag is checked: if set to RIP_RFC_METRIC COMPLIANT, the metric is sent as it is in the RIP database.

Cisco Metric Behavior

When the RIP process receives a response message, the metric-update flag is checked: if set to RIP_CISCO_METRIC COMPLIANT, it does not update the metric, but instead, uses the metric from the response message. When sending a response message, the metric-update flag is checked: if set to RIP_CISCO_METRIC COMPLIANT, and the route subtype is RIP_ROUTE_RTE, it increments the metric by adding the cost of the network and sends it to its neighbors.

Graceful Restart

Note: Graceful Restart applies only to RIP; RIPng does not support Graceful Restart.

There is no Internet-Draft for a similar RIP restart standard. Because RIP does not have a negotiation mechanism, it is difficult to add a restart negotiation mechanism similar to that used in BGP and OSPF.

In ZebOS-XP, the RIP process registers a preserve timer with NSM. NSM keeps the forwarding information until the preserve timer expires. After the RIP process restarts, it gets new routing information from neighboring routers, then installs new routing information into the NSM table. Newly installed information overwrites stale routes.

When some routes disappear during the restarting period, the route is marked as stale route, and is never overwritten by RIP. So, eventually the route is removed from FIB when the preserve timer is expired.

Use this command to force the restart of the RIP process:

```
(no) rip graceful-restart time <1-65535>
```

This command enables (the `no` parameter disables) RIP restart. <1-65535> is preserve time in seconds.

CHAPTER 2 Data Structures

This chapter provides information about files and their locations, data structures, and messaging.

Data Structures

- ripd
- ripng

ripd

struct rip

- General RIP structure
- Source file: ripd/ripd.h

Member	Definition
instance	RIP instance
flags	RIP flags
*rm	Pointer to RIP master
sock	RIP socket
*enable_network	RIP enabled network table
enable_if	RIP enabled interface vector
passive_if	RIP passive interface vector
*offset_list	RIP offset list
version	Default version of rip instance
*obuf	Output buffer of RIP
*if_table	RIP interface table
*nbr_table	RIP neighbor table
*table	RIP routing information base
*route	RIP only static routing information
*peer_table	RIP peer table
*t_read *t_update	RIP threads

Member	Definition
trigger *t_triggered_update *t_triggered_interval	Triggered update hack
clear_flags *t_clear_routes	RIP clear timer flags
update_time timeout_time garbage_time	RIP timer values
default_metric	RIP default metric
metric_type	RIP to enable/disable updating metric consistent to Cisco
default_information	RIP default-information originate
distance	RIP default distance
*distance_table	RIP source specific distance
distribute_master dm	RIP distribute list master
pcounter	RIP route counter in routing information base
pmax	Limit the number of routes
threshold	Percentage to give warning for maximum-prefix checking
recvbuf_size	RIP receive buffer size
redist	For redistribute
redist_map	Struct for redistribute route map

```
/* RIP structure. */
struct rip
{
  /* RIP instance. */
  int instance;

  /* RIP flags.  */
  u_char flags;
  #define                       (1 << 0)

  /* Pointer to RIP master.  */
  struct rip_master *rm;

  /* RIP socket. */
  int sock;

  /* RIP enabled network table. */
```



```
struct route_table *enable_network;

/* RIP enabled interface vector. */
vector enable_if;

/* RIP passive interface vector. */
vector passive_if;

/* RIP offset-list. */
struct list *offset_list;

/* Default version of rip instance. */
u_char version;

/* Output buffer of RIP. */
struct stream *obuf;

/* RIP interface table. */
struct route_table *if_table;

/* RIP neighbor table. */
struct route_table *nbr_table;

/* RIP routing information base. */
struct route_table *table;

/* RIP only static routing information. */
struct route_table *route;

/* RIP peer table. */
struct route_table *peer_table;

/* RIP threads. */
struct thread *t_read;
struct thread *t_update;

/* Triggered update hack. */
int trigger;
struct thread *t_triggered_update;
struct thread *t_triggered_interval;

/* RIP clear timer flags. */
u_int16_t clear_flags;
struct thread *t_clear_routes;

/* RIP timer values. */
u_int32_t update_time;
u_int32_t timeout_time;
u_int32_t garbage_time;
```

```
  /* RIP default metric. */
  int default_metric;

  /* RIP to enable/disable updating metric consistent to Cisco */
  u_char metric_type;

  /* RIP default-information originate. */
  u_char default_information;

  /* RIP default distance. */
  u_char distance;

  /* RIP source specific distance. */
  struct route_table *distance_table;

  /* RIP distribute list master. */
  struct distribute_master dm;

  /* RIP route counter in routing information base. */
  u_int32_t pcounter;

  /* Limit the number of routes. */
  u_int32_t pmax;

  /* Precentage to give warning for maximum-prefix checking */
  int threshold;

  /* RIP receive buffer size. */
  u_int32_t recvbuf_size;

  /* For redistribute.  */
  u_char redist[IPI_ROUTE_MAX];

  /* For redistribute route map. */
  struct
  {
    char *name;
    struct route_map *map;
    int metric_config;
    u_int32_t metric;
  } redist_map[IPI_ROUTE_MAX];
};
```

struct rip_master

- RIP master for system wide configuration and variables
- Source file: ripd/ripd.h

Member	Definition
*vr	Pointer to VR
zg	Pointer to globals
*rip	RIP instance list
config	RIP global configuration
flags	RIP global flags
*if_table	RIP global interface table
*if_params	RIP interface parameter pool
debug	RIP debug flags
global_route_changes	RIP route changes
global_queries	RIP queries
grace_period	RIP grace period

```
/* RIP master for system wide configuration and variables. */
struct rip_master
{
  /* Pointer to VR.  */
  struct ipi_vr *vr;

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

  /* RIP instance list.  */
  struct list *rip;

  /* RIP global configuration.  */
  u_char config;
#define RIP_GLOBAL_CONFIG_RESTART_GRACE_PERIOD  (1 << 0)

  /* RIP global flags.  */
  u_char flags;
#define RIP_GRACEFUL_RESTART                    (1 << 0)

  /* RIP global interface table.  */
  struct route_table *if_table;

  /* RIP interface parameter pool.  */
```

```
  struct list *if_params;

  /* RIP debug flags.  */
  struct
  {
    /* Debug flags for configuration.  */
    struct debug_rip conf;

    /* Debug flags for terminal.  */
    struct debug_rip term;

  } debug;

  /* RIP route changes.  */
  int global_route_changes;

  /* RIP queries.  */
  int global_queries;

#ifdef HAVE_RESTART
  /* RIP grace period.  */
  u_int32_t grace_period;
#endif /* HAVE_RESTART */
};
```

struct rip_if_param

- RIP interface configuration parameter
- Source file: ripd/rip_interface.h

Member	**Definition**
*ifname	Interface name
*rm	Pointer to RIP master
config	Configured flags
split_horizon	Split horizon type
recv_type;	Receive version type
send_type	Send version type
auth_mode	RIPv2 authentication mode
*auth_string	RIPv2 authentication string
*key_chain	RIPv2 authentication key-chain string

```
/* RIP interface configuration parameter. */
struct rip_if_params
```

```
{
  /* Interface name. */
  char *ifname;

  /* Pointer to RIP master.  */
  struct rip_master *rm;

  /* Configured flags. */
  u_int16_t config;
  #define RIP_IF_PARAM_ROUTER             (1 << 0)
  #define RIP_IF_PARAM_DISABLE_RECV       (1 << 1)
  #define RIP_IF_PARAM_DISABLE_SEND       (1 << 2)
  #define RIP_IF_PARAM_RECV_VERSION       (1 << 3)
  #define RIP_IF_PARAM_SEND_VERSION       (1 << 4)
  #define RIP_IF_PARAM_AUTH_MODE          (1 << 5)
  #define RIP_IF_PARAM_AUTH_STRING        (1 << 6)
  #define RIP_IF_PARAM_KEY_CHAIN          (1 << 7)
  #define RIP_IF_PARAM_SPLIT_HORIZON      (1 << 8)
  #define RIP_IF_PARAM_VERSION_1_USE      (1 << 9)

  /* Split horizon type. */
  u_char split_horizon;
  #define RIP_SPLIT_HORIZON_POISONED      0
  #define RIP_SPLIT_HORIZON_NONE          1
  #define RIP_SPLIT_HORIZON               2

  /* Receive and send version type. */
  u_char recv_type;
  u_char send_type;
  #define RI_RIP_UNSPEC                   0
  #define RI_RIP_VERSION_1                1
  #define RI_RIP_VERSION_2                2
  #define RI_RIP_VERSION_1_AND_2          3
  #define RI_RIP_VERSION_1_COMPATIBLE     4
  #define RI_RIP_VERSION_MAX              5

  /* RIPv2 authentication mode. */
  u_char auth_mode;
  #define RIP_NO_AUTH                     0
  #define RIP_AUTH_DATA                   1
  #define RIP_AUTH_SIMPLE_PASSWORD        2
  #define RIP_AUTH_MD5                    3

  /* RIPv2 authentication string. */
  char *auth_string;

  /* RIPv2 authentication key-chain string. */
  char *key_chain;
};
```

struct rip_offset_list

- RIP offset list parameters
- Source file - ripd/rip_offset.h

Member	Definition
*ifname	Interface name
direct	Access list and metric information

```
struct rip_offset_list
{
  char *ifname;

  struct
  {
    char *alist_name;
    u_char metric;
  } direct[RIP_OFFSET_LIST_MAX];
};
```

ripng

struct ripng

- RIPng structure
- Source file: ripngd/ripngd.h

Member	Definition
instance	RIPng instance
flags	RIPng flags
*rm	Pointer to RIPng master
sock	RIPng socket
*name	RIPng instance name
passive_if	RIPng passive interface vector
*offset_list	RIPng offset list
command	RIPng command
version	RIPng version
distance	RIPNG default distance

Member	Definition
*distance_table	RIPNG source specific distance
max_mtu	Max MTU
stream *ibuf stream *obuf	Input/output buffer of RIPng
*if_table	RIPng interface table
*nbr_table	RIPng neighbor table
*table	RIPng routing information base
*route	RIPng static route information
*aggregate	RIPng aggregate route information
*t_read *t_update	RIPng threads
trigger *t_triggered_update *t_triggered_interval	Triggered update threads
update_time timeout_time garbage_time	RIPng timer values
default_metric	RIPng default metric
default_information	RIPng default-information originate
metric_type	RIPng to enable/disable updating metric consistent to Cisco
recvbuf_size	RIPng receive buffer size
dm	RIPng distribute list master
ifrm	RIPng if rmap master
redist	For redistribute
redist_map	For redistribute route map

```
/* RIPng structure. */
struct ripng
{
  /* RIPng instance. */
  int instance;

  /* RIPng flags.  */
  u_char flags;
#define RIPNG_FLAG_UP                   (1 << 0)

  /* Pointer to RIPng master.  */
```

```
struct ripng_master *rm;

/* RIPng socket. */
int sock;

/* RIPng instance name. */
char *name;

/* RIPng passive interface vector. */
vector passive_if;

/* RIPng offset-list. */
struct list *offset_list;

/* RIPng Parameters.*/
u_char command;
u_char version;

/* RIPNG default distance. */
u_char distance;

/* RIPNG source specific distance. */
struct route_table *distance_table;

int max_mtu;

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

/* RIPng interface table. */
struct route_table *if_table;

/* RIPng neighbor table. */
struct route_table *nbr_table;

/* RIPng routing information base. */
struct route_table *table;

/* RIPng static route information. */
struct route_table *route;

/* RIPng aggregate route information. */
struct route_table *aggregate;

/* RIPng threads. */
struct thread *t_read;
struct thread *t_update;

/* Triggered update threads. */
```

```
  int trigger;
  struct thread *t_triggered_update;
  struct thread *t_triggered_interval;

  /* RIPng timer values. */
  u_int32_t update_time;
  u_int32_t timeout_time;
  u_int32_t garbage_time;

  /* RIPng default metric. */
  int default_metric;

  /* RIPng default-information originate. */
  int default_information;

  /* RIPng to enable/disable updating metric consistent to Cisco */
  u_char metric_type;

  /* RIPng receive buffer size. */
    u_int32_t recvbuf_size;

  /* RIPng distribute list master. */
  struct distribute_master dm;

  /* RIPng if rmap master. */
  struct if_rmap_master ifrm;

  /* For redistribute.  */
  u_char redist[IPI_ROUTE_MAX];

  /* For redistribute route map. */
  struct
  {
    char *name;
    struct route_map *map;
    int metric_config;
    u_int32_t metric;
  } redist_map[IPI_ROUTE_MAX];
};
```

struct ripng_master

- RIPng master of system wide configuration and variables
- Source file - ripngd/ripngd.h

Attribute	Definition
*vr	Pointer to VR
*zg	Pointer to globals
*ripng	RIPng instance list
*if_table	RIPng global interface table
*if_params	RIPng interface parameter pool
debug	RIPng debug flags

```
/* RIPng master of system wide configuration and variables. */
struct ripng_master
{
  /* Pointer to VR.  */
  struct ipi_vr *vr;

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

  /* RIPng instance list.  */
  struct list *ripng;

  /* RIPng global interface table.  */
  struct route_table *if_table;

  /* RIPng interface parameter pool.  */
  struct list *if_params;

  /* RIPng debug flags.  */
  struct
  {
    /* Debug flags for configuration. */
    struct debug_ripng conf;

    /* Debug flags for terminal. */
    struct debug_ripng term;

  } debug;
};
```

struct ripng_info

- Each route's information
- Source file - ripngd/ripngd.h

Member	Definition
ripng *top	Pointer to the parent
*rn	Back pointer to the route node
type	This route type: static, ripng or aggregate
sub_type	Sub type for static route
nexthop from	RIPng specific information
ifindex	Which interface this route is from
metric	Metric of this route
tag	Tag field of RIPng packet
suppress	For aggregation
flags	Flags of RIPng route
*t_timeout *t_garbage_collect timeout_start_time	Garbage collect timer
metric_set	Route-map features - this variables can be changed
distance	Distance

```
/* Each route's information. */
struct ripng_info
{
  /* Parent's pointer. */
  struct ripng *top;

  /* Back pointer to the route node. */
  struct route_node *rn;

  /* This route's type.  Static, ripng or aggregate. */
  u_char type;

  /* Sub type for static route. */
  u_char sub_type;

  /* RIPng specific information */
  struct pal_in6_addr nexthop;
  struct pal_in6_addr from;
```

```
  /* Which interface does this route come from. */
  unsigned int ifindex;

  /* Metric of this route.  */
  u_char metric;

  /* Tag field of RIPng packet.*/
  u_int16_t tag;

  /* For aggregation. */
  unsigned int suppress;

  /* Flags of RIPng route. */
  #define RIPNG_RTF_FIB      1
  #define RIPNG_RTF_CHANGED  2
  u_char flags;

  /* Garbage collect timer. */
  struct thread *t_timeout;
  struct thread *t_garbage_collect;
  struct pal_timeval timeout_start_time;

  /* Route-map features - this variables can be changed. */
  u_char metric_set;

  /* Distance.  */
  u_char distance;
};
```

struct ripng_aggregate

- Aggregated routes information
- Source file: ripngd/ripng_route.h

Member	Definition
count	Aggregate route count
suppress	Suppressed route count
metric	Metric of this route
tag	Tag field of RIPng packet

```
struct ripng_aggregate
{
  /* Aggregate route count. */
  unsigned int count;

  /* Suppressed route count. */
  unsigned int suppress;
```

```
  /* Metric of this route.  */
  u_char metric;

  /* Tag field of RIPng packet.*/
  u_int16_t tag;
};
```

struct ripng_if_param

- RIPng interface configuration parameter
- Source file - ripngd/ripng_interface.h

Member	Definition
*ifname	Interface name
ripng_master *rm	Pointer to RIPng master
config	Configured flags
*name	RIPng instance name
split_horizon	Split horizon type

```
/* RIPng interface configuration parameter. */
struct ripng_if_params
{
  /* Interface name. */
  char *ifname;

  /* Pointer to RIPng master.  */
  struct ripng_master *rm;

  /* Configured flags. */
  u_char config;
  #define RIPNG_IF_PARAM_ROUTER           (1 << 0)
  #define RIPNG_IF_PARAM_SPLIT_HORIZON    (1 << 1)

  /* RIPng instance name. */
  char *name;

  /* Split horizon type. */
  u_char split_horizon;
  #define RIPNG_SPLIT_HORIZON_POISONED     0
  #define RIPNG_SPLIT_HORIZON_NONE         1
  #define RIPNG_SPLIT_HORIZON              2
};
```

struct ripng_offset_list

- RIPng offset list parameters
- Source file - ripngd/ripng_offset.h

Member	Definition
*ifname	Interface name
direct	Access list information

```
struct ripng_offset_list
{
  char *ifname;

  struct
  {
    char *alist_name;
    u_char metric;
  } direct[RIPNG_OFFSET_LIST_MAX];
};
```

CHAPTER 3 RIP Command API

This chapter contains Command Line Interface functions for Routing Information Protocol (RIP).

Note: The `vr_id` parameter in these functions support Virtual Routers (VRs). For an implementation without Virtual Routers, you must pass value 0 for the VR ID parameter. This is the default value, known as PVR (Privileged VR) ID.

RIP APIs

The table below is a summary of the RIPng functions. Details are provided in the following subsections

Function	Description
rip_cisco_metric_behavior_set	Updating the metric consistent with Cisco is enabled
rip_cisco_metric_behavior_unset	Reverts to the default metric type: RIP RFC compliant
rip_default_metric_set	Sets the routing protocol to use the specified metric value for all redistributed routes
rip_default_metric_unset	Unsets the metrics assigned to redistributed routes
rip_distance_set	Specifies the administrative distance for the route calculation
rip_distance_set_default	Sets the administrative distance to the specified value
rip_distance_unset_default	Resets the administrative distance configuration
rip_distance_unset	Deletes the entry (distance) from the table
rip_enable_if_add	Enables RIP routing on the specified interface
rip_enable_if_delete	Disables RIP routing on the specified interface
rip_enable_nbr_add	Enables RIP routing on the specified neighbor
rip_enable_nbr_delete	Disables RIP routing on the neighbor
rip_enable_network_add	Enables RIP routing on the specified network
rip_enable_network_delete	Disables RIP routing on the specified network
rip_if_auth_key_set	Specifies the RIP authentication key chain string
rip_if_auth_key_unset	Disables key chain authentication
rip_if_auth_mode_set	Specifies the type of authentication mode used for RIP v2 packets
rip_if_auth_mode_unset	Resets the authentication mode
rip_if_auth_str_set	Sets the authentication string or password used by a key

Function	Description
rip_if_auth_str_unset	Disables receiving RIP packets on specified interface
rip_if_receive_packet_set	Enables the interface to receive RIP packets
rip_if_receive_packet_unset	Disables receiving RIP packets on the specified interface
rip_if_receive_version_type_set	Enables receiving the specified version(s) of RIP packets
rip_if_receive_version_unset	Resets the receive version to the RIP node version
rip_if_send_packet_set	Enables sending RIP packets through the current interface
rip_if_send_packet_unset	Disables sending RIP packets on the specified interface
rip_if_send_version_type_set	Sets sending RIP packets on an interface using version control
rip_if_send_version_unset	Sets the sending version to the version of the RIP node
rip_if_split_horizon_set	Enables RIP split-horizon behavior
rip_if_split_horizon_poisoned_set	Enables RIP split-horizon poisoned reverse behavior
rip_if_split_horizon_unset	Disables split horizon behavior
rip_instance_set	Establishes an instance of the RIP router
rip_instance_unset	Removes an instance of the RIP router
rip_max_route_set	Sets the maximum prefix
rip_max_route_unset	Sets the threshold value to the default threshold percentage
rip_offset_list_set	Adds an offset to in and out metrics to routes learned through RIP
rip_offset_list_unset	Removes the offset list
rip_passive_if_add	Suppresses RIP updates
rip_passive_if_delete	Disables blocking RIP broadcasts on the interface
rip_recvbuf_size_set	Specifies the size of the RIP UDP buffer
rip_recvbuf_size_unset	Resets the size of the RIP UDP buffer
rip_redistribute_metric_rmap_set	Specifies the metric of the route map
rip_redistribute_metric_set	Redistributes information from other routing protocols
rip_redistribute_rmap_set	Redistributes information from other routing protocols
rip_redistribute_set	Redistributes information from other routing protocols
rip_redistribute_unset	Resets the learned routes
rip_restart_set	Restore the RIP routes in the NSM routing table and forces the RIP process to restart

Function	Description
rip_restart_grace_period_set	Sets the grace period of RIP graceful restart
rip_restart_grace_period_unset	Resets the grace period to RIP_RESTART_GRACE_PERIOD_DEFAULT
rip_route_add	Sets static RIP routes
rip_route_default_add	Generates a default route into the RIP
rip_route_default_delete	Disables the configuration of a default route
rip_route_delete	Removes the specified static route
rip_route_type_delete	Clears specific data from the RIP routing table
rip_timers_set	Sets the specified time per RIP timer
rip_timers_unset	Resets the three timers to the default values
rip_version_set	Sets the version of the RIP node
rip_version_unset	Resets the version of the RIP node

rip_cisco_metric_behavior_set

This function sets the metric update as Cisco; updating the metric consistent with Cisco is enabled. By default, Cisco metric-behavior is disabled.

This function is called by the following command:

```
cisco-metric-behavior
```

Syntax

```
int
rip_cisco_metric_behavior_set (u_int32_t vr_id, int instance, u_charmetric_type);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`metric_type`	The metric type: Cisco behavior or RFC-compliant behavior. The default value: RIP RFC compliant

Output Parameters

None

Return Values

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_METRIC_TYPE_INVALID when `metric_type` is greater than RIP_CISCO_METRIC_COMPLIANT

RIP_API_SET_SUCCESS when the call is successful

rip_cisco_metric_behavior_unset

This function unsets updating the metric consistent with Cisco and reverts to the default metric type: RIP RFC compliant.

This function is called by the following command:

```
no cisco-metric-behavior
```

Syntax

```
int
rip_cisco_metric_behavior_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_SUCCESS when the call is successful

rip_default_metric_set

This function sets the routing protocol to use the specified metric value for all redistributed routes. The specified default metric will be used by all routes that are redistributed.

This function is called by the following command:

```
default-metric <1-16>
```

Syntax

```
int
rip_default_metric_set (u_int32_t vr_id, int instance, int metric);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`instance`	The number of the instance
`metric`	The metric of the offset list

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_METRIC_INVALID when the `metric_str` is NULL

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_default_metric_unset

This function resets the metrics assigned to redistributed routes to the default setting: 1.

This function is called by the following command:

```
no default-metric <1-16>
```

Syntax

```
int
rip_default_metric_unset (u_int32_t vr_id, int rip_instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`rip_instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_distance_set

This function specifies the administrative distance for the route calculation. The distance is a feature used by the routers to select the path when there are two or more different routes to the same destination from two different routing protocols.

A smaller administrative distance indicates a more reliable protocol.

This function is called by the following command:

```
distance A.B.C.D/M
```

Syntax

```
int
rip_distance_set (u_int32_t vr_id, int instance, char *distance_str,
struct pal_in4_addr *addr, int plen, char *alist);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`rip_instance`	
	The number of the instance
`distance_str`	The distance value
`addr`	The address of the source prefix
`plen`	The prefix length for the static RIP route
`char *alist`	The access list

Output Parameters

None

Return Values

RIP_API_SET_ERR_DISTANCE_INVALID when distance entered is invalid

RIP_API_SET_ERR_INVALID_VALUE when the `distance_str` is NULL

RIP_API_SET_ERR_PREFIX_INVALID when prefix entered is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_distance_set_default

This function sets the administrative distance to the specified value: 1–255.

This function is called by the following command:

```
distance <1-255>
```

Syntax

```
int
rip_distance_set_default (u_int32_t vr_id, int instance, char *distance_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`distance_str`	The pointer to distance value

Output Parameters

None

Return Values

RIP_API_SET_ERR_DISTANCE_INVALID when distance entered is invalid

RIP_API_SET_ERR_INVALID_VALUE when the `distance_str` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_distance_unset_default

This function resets the administrative distance configuration to the default value: 120.

This function is called by the following command:

```
no distance <1-255>
```

Syntax

```
int
rip_distance_unset_default (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_distance_unset

This function deletes the administrative distance that was configured for the route calculation.

This function is called by the following command:

```
no distance <1-255> A.B.C.D/M (WORD)
```

Syntax

```
int
rip_distance_unset (u_int32_t vr_id, int instance, struct pal_in4_addr *addr,
int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The address of the source prefix
`plen`	The prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_DISTANCE_NOT_EXIST when the entry cannot be found in the table

RIP_API_SET_ERR_INVALID_VALUE when the `distance_str` is NULL

RIP_API_SET_ERR_PREFIX_INVALID when prefix entered is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_if_add

This function enables RIP routing on the specified interface. If a network is not specified, the interfaces in that network will not be advertised in any RIP update.

This function is called by the following command:

```
network IFNAME
```

Syntax

```
int
rip_enable_if_add (u_int32_t vr_id, int instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`ifname`	The interface name for which RIP routing is enabled

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_EXIST when routing is already enabled for this interface

RIP_API_SET_ERR_INVALID_VALUE when the interface name is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_if_delete

This function disables RIP routing on the specified interface.

This function is called by the following command:

```
no network IFNAME
```

Syntax

```
int
rip_enable_if_delete (u_int32_t vr_id, int instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_ NOT_EXIST when interface cannot be found

RIP_API_SET_ERR_INVALID_VALUE when the interface name is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_nbr_add

This function enables RIP routing on the specified neighbor; RIP updates are sent to the unicast IP address(es) specified in the neighbor statement(s).

- Multiple neighbor commands can be used to specify additional neighbors or peers.
- This command permits point-to-point (nonbroadcast) exchange of routing information.

This function is called by the following command:

```
neighbor A.B.C.D
```

Syntax

```
int
rip_enable_nbr_add (u_int32_t vr_id, int instance, struct pal_in4_addr *addr)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The neighbor address on which this call enables RIP routing

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `neighbor_prefix` is NULL

RIP_API_SET_ERR_NBR_STATIC_EXIST when neighbor is already in table

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_nbr_delete

This function disables RIP routing on the neighbor.

This function is called by the following command:

```
no neighbor A.B.C.D
```

Syntax

```
int
rip_enable_nbr_delete (u_int32_t vr_id, int instance, struct pal_in4_addr *addr);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The neighbor address on which this call enables RIP routing

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `neighbor_prefix` is NULL

RIP_API_SET_ERR_NBR_STATIC_EXIST when neighbor is already in table

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_network_add

This function enables RIP routing on the specified network: specifies a network as one that runs Routing Information Protocol (RIP).

- This command specifies the networks to which routing updates will be sent and received.
- If a network is not specified, the interfaces in that network will not be advertised in any RIP update.

This function is called by the following command:

```
network A.B.C.D/M
```

Syntax

```
int
```

```
rip_enable_network_add (u_int32_t vr_id, int instance,
struct pal_in4_addr *addr, int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The network address on which this call enables RIP routing
`plen`	The prefix length for the enabled network

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the value is neither 1 nor 2

RIP_API_SET_ERR_NETWORK_EXIST when routing is already enabled for this network

RIP_API_SET_ERR_PREFIX_INVALID when prefix entered is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_enable_network_delete

This function disables RIP routing on the specified network.

This function is called by the following command:

```
no network A.B.C.D/M
```

Syntax

```
int
rip_enable_network_delete (u_int32_t vr_id, int instance,
struct pal_in4_addr *addr, int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The network address on which this call enables RIP routing
`plen`	The prefix length for the enabled network

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the value is neither 1 nor 2

RIP_API_SET_ERR_NETWORK _NOT_EXIST when routing is not enabled for this network

RIP_API_SET_ERR_PREFIX_INVALID when prefix entered is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_key_set

This function specifies the RIP authentication key chain string: enable RIPv2 authentication on an interface and specify the name of the key chain to be used. Not configuring a key chain results in no authentication.

This function is called by the following command:

```
ip rip authentication key-chain LINE
```

Syntax

```
int
rip_if_auth_key_set (u_int32_t vr_id, char *ifname, char *str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name
`str`	The authentication key chain string

Output Parameters

None

Return Values

RIP_API_SET_ERR_AUTH_STR_EXIST when the authentication string already exists

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_key_unset

This function clears the key chain authentication; authentication is disabled.

This function is called by the following command:

```
no ip rip authentication key-chain
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_mode_set

This function sets the authentication mode: specify the type of authentication mode used for RIP v2 packets.

This function is called by the following command:

```
ip rip authentication mode
```

Syntax

```
int
rip_if_auth_mode_set (u_int32_t vr_id, char *ifname, char *str);
```

Input Parameters

`vr_id`		The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`		The interface name
`str`		The specified authentication mode:
	`text`	Plain text authentication (default mode)
	`md5`	MD5 authentication

Output Parameters

None

Return Values

RIP_API_SET_ERR_AUTH_TYPE_INVALID when auth_mode_str is neither “md5” nor “text”. (VxWorks does not support MD5 authentication. See `rip_api.c`.)

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_mode_unset

This function resets the authentication mode.

- If the authentication string or key chain exist, the mode is set to plain text authentication.
- If no mode is specified, the mode is set to no authentication.

This function is called by the following command:

```
no ip rip authentication mode
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_str_set

This function sets the authentication string or password used by a key.

- Use this command to specify the password for a single key on an interface.

This function is called by the following command:

```
ip rip authentication string
```

Syntax

```
int
rip_if_auth_str_set (u_int32_t vr_id, char *ifname, char *str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name from the `interface IFNAME` CLI command
`str`	The authentication string from the command. It is either a text string or an MD5 string

Output Parameters

None

Return Values

Note: The VTY displays the foll message when the authentication string is longer than 16 characters: “RIPv2 authentication string must be shorter than 16”

RIP_API_SET_ERR_KEYCHAIN_EXIST when the key chain already has a value

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_auth_str_unset

This function disables receiving RIP packets on specified interface.

This function is called by the following command:

```
no ip rip authentication string
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_receive_packet_set

This function enables the interface to receive RIP packets. This is the default setting.

This function is called by the following command:

```
ip rip receive-packet
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name string

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_receive_packet_unset

This function disables receiving RIP packets on the specified interface.

This function is called by the following command:

```
no ip rip receive-packet
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name string

Output Parameters

None

Return Values

This function returns:

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_receive_version_type_set

This function enables receiving the specified version of RIP packets (version 1 or version 2) or receiving both versions of RIP packets (version 1 and version 2).

This function is called by the following CLI commands:

```
ip rip receive version (1|2)
ip rip receive version 1 2
```

Syntax

```
int
rip_if_receive_version_type_set (u_int32_t vr_id, char *ifname, int type);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`char`	`*ifname` The interface name string
`int type`	The version type, one of the following:

`RI_RIP_VERSION_1`

RIP version 1

`RI_RIP_VERSION_2`

RIP version 2

`RI_RIP_VERSION_1_AND_2`

RIP version 1 and 2

`RI_RIP_VERSION_1_COMPATIBLE`

RIP version 2 with uni/broadcast address

Output Parameters

None

Return Values

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

Usage Notes

The command:

```
!
ip rip receive version 1 2
```

The Syntax:

```
int retval;
retval= rip_if_receive_version_type_set (ifname, RI_RIP_VERSION_1_AND_2);
```

rip_if_receive_version_unset

This function resets the receive version to the RIP node version.

This function is called by the following command:

```
no ip rip receive version
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_send_packet_set

This function enables sending RIP packets through the current interface.

This function is called by the following command:

```
ip rip send-packet
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name string

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_send_packet_unset

This function disables sending RIP packets on the specified interface.

This function is called by the following command:

```
no ip rip send-packet
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name string

Output Parameters

None

Return Values

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_send_version_type_set

This function sets sending RIP packets on an interface using version control (version type).

This function is called by the following command:

`ip rip send version` (1|2|1-compatible)

Syntax

```
int
rip_if_send_version_type_set (u_int32_t vr_id, char *ifname, int type);
```

Input Parameters

`vr_id` The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.

`ifname` The interface name.

`int type` The version type, one of the following:

`RI_RIP_VERSION_1`
RIP version 1

`RI_RIP_VERSION_2`
RIP version 2

`RI_RIP_VERSION_1_AND_2`
RIP version 1 and 2

`RI_RIP_VERSION_1_COMPATIBLE`
RIP version 2 with uni/broadcast address

Output Parameters

None

Return Values

RIP_API_SET_ERR_VERSION_INVALID when the version is outside the range RI_RIP_UNSPEC to RI_RIP_VERSION_MAX specified in `ripd.h`

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

Usage Notes

The command:

```
!
ip rip send version 1 2
```

The Syntax:

```
int retval;
retval= rip_if_send_version_type_set (ifname, RI_RIP_VERSION_1_AND_2);
```

rip_if_send_version_unset

This function sets the sending version to the version of the RIP node, the default value.

This function is called by the following command:

```
no ip rip send version
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_SUCCESS when the call is successful

rip_if_split_horizon_set

This function enables RIP split-horizon behavior. This command helps avoid including routes in updates sent to the same gateway from which they were learned.

- Using the split horizon command omits routes learned from one neighbor, in updates sent to that neighbor.
- The default configuration is split-horizon poisoned.

This function is called by the following command:

```
ip rip split-horizon
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_SPLIT_HORIZON_INVALID when split-horizon type is invalid

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_split_horizon_poisoned_set

This function enables RIP split-horizon poisoned reverse behavior.

- Using the poisoned parameter with this command includes such routes in updates, but sets their metrics to infinity: advertising that these routes are not reachable.
- The default configuration is split-horizon poisoned.

This function is called by the following CLI commands:

```
ip rip split-horizon poisoned
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when interface parameters are not configured

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_if_split_horizon_unset

This function disables split horizon behavior. The default configuration is split-horizon poisoned.

This function is called by the following command:

```
no ip rip split-horizon
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_SPLIT_HORIZON_INVALID when split-horizon type is invalid

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_instance_set

This function establishes an instance of the RIP router; the RIP routing process is enabled.

This function is called by the following command:

```
router rip
```

Syntax

```
int
rip_instance_set (u_int32_t vr_id, int instance)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_instance_unset

This function removes an instance of the RIP router: disable the RIP routing process.

This function is called by the following command:

```
no router rip
```

Syntax

```
int
rip_instance_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_max_route_set

This function sets the maximum number of RIP routes that can be stored in the routing table. It also sets the percentage of maximum routes to generate a warning (default maximum 75%).

This function is called by the following command:

```
maximum-prefix
```

Syntax

```
int
rip_max_route_set (u_int32_t vr_id, int instance, char *pmax_str,
char *threshold_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`pmax_str`	The maximum prefix <1–65535.
`threshold_str`	The threshold value <1–100>; default value is 75

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `maximum_prefix_str` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found.

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_max_route_unset

This function sets the threshold value to the default threshold percentage of maximum-prefix checking: RIP_ROUTE_THRESHOLD_DEFAULT (see ripd.h). The default threshold percentage is 75%.

This function is called by the following command:

```
no maximum-prefix
```

Syntax

```
int
rip_max_route_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_offset_list_set

This function adds an offset to in and out metrics to routes learned through RIP: specifies the offset value that is added to the routing metric.

- When the networks match the access list, the offset is applied to the metrics.
- No change occurs when the offset value is zero.

This function is called by the following command:

`offset-list`

Syntax

```
int
rip_offset_list_set (u_int32_t vr_id, int instance, char *alist,
char *direct_str, int metric, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`alist`	The access list (alist) name
`direct_str`	The string of “in” or “out”
`metric`	The metric of the offset, range `0-RIP_METRIC_INFINITY`
`ifname`	The name of the interface

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `alist` is NULL

RIP_API_SET_ERR_METRIC_INVALID when the `metric` is outside the `0-RIP_METRIC_INFINITY` range (see `ripd.h`)

RIP_API_SET_ERR_OFFSET_LIST_NOT_EXIST when the offset list cannot be found

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the interface instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_offset_list_unset

This function removes the offset list.

This function is called by the following command:

```
no offset-list
```

Syntax

```
int
rip_offset_list_unset (u_int32_t vr_id, int instance, char *alist,
char *direct_str, int metric, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`char *alist`	The access list name
`char *direct_str`	The string of “in” or “out”
`int metric`	The metric of the offset, set in the range of `0-RIP_METRIC_INFINITY`
`ifname`	The name of the interface

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `access_list_str` is NULL

RIP_API_SET_ERR_METRIC_INVALID when the `metric` is outside the `0-RIP_METRIC_INFINITY` range (see `ripd.h`)

RIP_API_SET_ERR_OFFSET_LIST_NOT_EXIST when the offset list cannot be found

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the interface instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_passive_if_add

This function suppresses RIP updates: blocks RIP broadcast on the interface.

This function is called by the following command:

```
passive-interface
```

Syntax

```
int
rip_passive_if_add (u_int32_t vr_id, int instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_EXIST when routing is already enabled for this interface

RIP_API_SET_ERR_INVALID_VALUE when the interface name is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_passive_if_delete

This function disables blocking RIP broadcasts on the interface.

This function is called by the following command:

```
no passive-interface
```

Syntax

```
int
rip_passive_if_delete (u_int32_t vr_id, int rip_instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`int rip_instance`	The number of the instance
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when routing is not enabled for this interface

RIP_API_SET_ERR_INVALID_VALUE when the interface name is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_recvbuf_size_set

This function specifies the size of the RIP UDP buffer.

This function is called by the following command:

```
recv-buffer-size
```

Syntax

```
int
rip_recvbuf_size_set (u_int32_t vr_id, int instance, u_int32_t bufsize);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`bufsize`	The size of the receiving buffer <8192–2147483647>

Output Parameters

None

Return Values

RIP_API_SET_ERR_CANT_CHANGE_BUFFER_SIZE when the buffer size cannot be set by the lower layer, i.e., kernel

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_recvbuf_size_unset

This function resets the size of the RIP UDP buffer to the default value: (1024*192).

This function is called by the following command:

```
no recv-buffer-size
```

Syntax

```
int
rip_recvbuf_size_set (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_CANT_CHANGE_BUFFER_SIZE when the buffer size cannot be set by the lower layer, such as kernel

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_redistribute_metric_rmap_set

This call function specifies the metric of the route map.

Syntax

```
int
rip_redistribute_metric_rmap_set (u_int32_t vr_id, int instance, char *type_str,
int metric, char *name)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`type_str`	The route type
`metric`	The metric value
`name`	The route map name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the type_str is NULL

RIP_API_SET_ERR_METRIC_INVALID when the metric is not in the range 0-RIP_METRIC_INFINITY

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_redistribute_metric_set

This function redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIP. It also redistributes kernel, connected and static into the RIP. The configured metric is set to the redistributing routes.

This function is called by the following command:

```
redistribute metric
```

Syntax

```
int
rip_redistribute_metric_set (u_int32_t vr_id, int instance, char *type_str,
int metric);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`instance`	The number of the instance
`type_str`	The route type
`metric`	The metric value <0–16>

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the type_str is NULL

RIP_API_SET_ERR_METRIC_INVALID when the metric is not in the range 0-RIP_METRIC_INFINITY

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_redistribute_rmap_set

This function redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIP. It also redistributes kernel, connected and static into the RIP. Route redistribution is set per route map.

This function redistributes information from other routing protocols.

This function is called by the following command:

```
redistribute route-map
```

Syntax

```
int
rip_redistribute_rmap_set (u_int32_t vr_id, int instance, char *type_str,
char *name);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`type_str`	The route type: `kernel`, `connected`, `static`, `ospf`, `isis` or `bgp`
`name`	The route map name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `name` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_redistribute_set

This function redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIP. It also redistributes kernel, connected and static into the RIP information

This function is called by the following command:

```
redistribute
```

Syntax

```
int
rip_redistribute_set (u_int32_t vr_id, int instance, char *type_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`char *type_str`	The route type: `kernel`, `connected`, `static`, `ospf`, `isis` or `bgp`

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the type_str (route type) is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_redistribute_unset

This function resets the learned routes.

This function is called by the following command:

```
no redistribute
```

Syntax

```
int
rip_redistribute_unset (u_int32_t vr_id, int instance, char *type_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`type_str`	The route type: `kernel`, `connected`, `static`, `ospf`, `isis` or `bgp`

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the type_str (route type) is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_restart_set

This function preserves NSM to restore the RIP routes in the NSM routing table and forces the RIP process to restart.

- After this function executes, the router immediately shuts down.
- It notifies NSM that RIP has shut down gracefully and NSM preserves routes installed by RIP until the grace-period expires.

This function is called by the following command:

```
restart rip graceful
```

Note: This command is available only when the configuration option `--enable-restart` is enabled when compiling ZebOS-XP.

Syntax

```
int
rip_restart_set (u_int32_t vr_id, u_int32_t seconds);
```

Input parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`seconds`	The RIP restart grace period in seconds; default is 65535

Output parameters

None

Return Values

RIP_API_SET_ERR_GRACE_PERIOD_INVALID when the value of the RIP restart grace period is invalid

RIP_API_SET_ERR_VR_NOT_EXIST when an invalid Virtual Router ID is passed

RIP_API_SET_SUCCESS when the call is successful

rip_restart_grace_period_set

This function sets the grace period of RIP graceful restart.

- NSM is notified about the Grace Period.
- If the RIP daemon unexpectedly shuts down, NSM sends this value to the RIP daemon when it comes up again, and the RIP daemon uses this value to end the Graceful state.

This function is called by the following command:

```
rip restart grace-period
```

Syntax

```
int
rip_restart_grace_period_set (u_int32_t vr_id, u_int32_t seconds);
```

Input parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`seconds`	The grace period in seconds <1-65535>

Output parameters

None

Return Values

RIP_API_SET_ERR_GRACE_PERIOD_INVALID when the value of the RIP restart grace period is invalid

RIP_API_SET_ERR_VR_NOT_EXIST when The Virtual Router ID does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_restart_grace_period_unset

This function resets the grace period seconds to its default constant value: 65535.

This function is called by the following command:

```
no rip restart grace-period
```

Syntax

```
int
rip_restart_grace_period_unset (u_int32_t vr_id);
```

Input parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.

Output parameters

None

Return value

RIP_API_SET_ERR_VR_NOT_EXIST when The Virtual Router ID does not exist

RIP_API_SET_SUCCESS when the call is success

rip_route_add

This function configures a static route for advertisement through RIP explicitly. An ideal configuration includes a static route that is redistribute via redistribute static inside a routing process.This command eliminates that overhead; efficient for debug.

This function is called by the following command:

```
route
```

Syntax

```
int
rip_route_add (u_int32_t vr_id, int instance, struct pal_in4_addr *addr,
int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The static RIP route address
`plen`	The prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `neighbor_prefix` is NULL

RIP_API_SET_ERR_NETWORK_EXIST when network is already in the table

RIP_API_SET_ERR_PREFIX_INVALID when prefix entered is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_route_default_add

This function generates a default route into the Routing Information Protocol (RIP).

This function is called by the following command:

```
default-information
```

Syntax

```
int
rip_route_default_add (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found.

RIP_API_SET_ERR_NETWORK_EXIST when network is already in the table

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_route_default_delete

This function disables the configuration of a default route into the Routing Information Protocol (RIP).

This function is called by the following command:

```
no default-information
```

Syntax

```
int
rip_route_default_delete (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_NETWORK_EXIST when network is already in the table

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_route_delete

This function removes the specified static route.

This function is called by the following command:

```
no route
```

Syntax

```
int
rip_route_delete (u_int32_t vr_id, int instance, struct pal_in4_addr *addr,
int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`addr`	The static RIP route address
`plen`	Prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `neighbor_prefix` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_NETWORK_NOT_EXIST when network is not in the table

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_route_type_delete

This function clears specific data from the RIP routing table.

This function is called by the following command:

```
clear ip rip route
```

Syntax

```
int
rip_route_type_delete (u_int32_t vr_id, int instance, char *str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`str`	The route type: `kernel`, `static`, `connected`, `rip`, `ospf`, `isis`, `bgp`, or `all`

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the `str` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_ROUTE_NOT_EXIST when route cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_timers_set

This function sets the specified time per RIP timer: update timer; timeout timer; garbage timer.

- At the specified interval, the update timer sends an update containing the complete routing table to every neighboring router. When the time specified by the timeout parameter expires the route is no longer valid.
- For a short period, the routing information is retained in the routing table so that neighbors are notified that the route has been dropped.The route is included in all updates until the specified garbage time expires.

All routers in the network must have the same timers to allow RIP to execute a distributed and asynchronous routing algorithms. The timers should not be synchronized as that might lead to unnecessary collisions on the network.

This function is called by the following command:

```
timer
```

Syntax

```
int
rip_timers_set (u_int32_t vr_id, int instance, u_int32_t update,
u_int32_t timeout, u_int32_t garbage);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`instance`	The number of the instance
`update`	The number of update timer seconds
`timeout`	The number of timeout timer seconds
`garbage`	The number of garbage timer seconds

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_timers_unset

This call resets the three timers to the default values:

- Update timer, 30 seconds
- Timeout timer, 180 seconds
- Garbage timer, 120 seconds

This function is called by the following command:

```
no timer
```

Syntax

```
int
rip_timers_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance.

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_version_set

This function resets the RIP version to RIPv2 that is globally used by the router.

This function is called by the following command:

```
no timer
```

Syntax

```
int
rip_version_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

rip_version_unset

This function resets the RIP version to RIPv2 that is globally used by the router.

This function is called by the following command:

```
no timer
```

Syntax

```
int
rip_version_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when vr_id does not exist

RIP_API_SET_SUCCESS when the call is successful

CHAPTER 4 RIPng Command API

This chapter contains the command AP for IPv6 Routing Information Protocol (RIPng).

Note: The `vr_id` parameter in these functions support Virtual Routers (VRs). For an implementation without Virtual Routers, you must pass value 0 for the VR ID parameter. This is the default value, known as PVR (Privileged VR) ID.

RIPng API Functions

The table below is a summary of the RIPng functions. Details are provided in the following subsections.

Function	Description
ripng_aggregate_add	Aggregates RIP route announcements
ripng_aggregate_delete	Deletes aggregate advertising routes
ripng_cisco_metric_behavior_set	Sets the metric update as Cisco
ripng_cisco_metric_behavior_unset	Unsets the metric update as Cisco
ripng_default_metric_set	Sets the routing protocol to use the specified metric value for all redistributed routes
ripng_default_metric_unset	Unsets the metrics assigned to redistributed routes
ripng_distance_set_default	Sets the administrative distance to the specified value
ripng_distance_unset_default	Resets the administrative distance to its default value:120
ripng_enable_nbr_add	Enables RIP routing on the specified neighbor
ripng_enable_nbr_delete	Disables RIP routing on the neighbor
ripng_if_ipv6_router_set	Enables RIPng routing on the interface
ripng_if_ipv6_router_unset	Disables RIPng routing on the interface
ripng_if_split_horizon_poisoned_set	Enables RIP split-horizon poisoned reverse behavior
ripng_if_split_horizon_set	Enables the RIP split-horizon behavior
ripng_if_split_horizon_unset	Disables the split horizon behavior
ripng_instance_set	Establishes an instance of the ipV6 RIP router
ripng_instance_unset	Removes an instance of the IPv6 router
ripng_offset_list_set	Adds an offset to in and out metrics to routes learned through RIP
ripng_offset_list_unset	Removes the offset list
ripng_passive_if_add	Suppresses RIP updates

Function	Description
ripng_passive_if_delete	Disables blocking RIP broadcasts on the interface
ripng_recvbuf_size_set	Sets the specified size of the RIP UDP buffer
ripng_recvbuf_size_unset	Resets the size of the RIP UDP buffer
ripng_redistribute_metric_set	Redistributes information from other routing protocols
ripng_redistribute_metric_rmap_set	Redistributes information from other routing protocols
ripng_redistribute_rmap_set	Redistributes information from other routing protocols
ripng_redistribute_set	Redistributes information from other routing protocols
ripng_redistribute_unset	Unsets the redistribute configuration
ripng_route_add	Configures static RIP routes
ripng_route_default_add	Generates a default route into the RIP
ripng_route_default_delete	Disables the configuration of the default route into the RIP
ripng_route_delete	Unsets the configured RIPng static route
ripng_route_type_delete	Clears specified data from the RIPng routing table
ripng_timers_set	Sets the specified time per RIP timer
ripng_timers_unset	Resets the three timers to the default values

ripng_aggregate_add

This function aggregates RIP routes. A RIPng router announces a route to the aggregated prefix with a metric of 1.

This function is called by the following command:

```
aggregate-address
```

Syntax

```
int
ripng_aggregate_add (u_int32_t vr_id, int rip_instance,
struct pal_in6_addr *addr, int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`rip_instance`	The instance number
`addr`	The aggregated route prefix
`plen`	The prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the address is invalid

RIP_API_SET_ERR_NETWORK_EXIST when the network already exists

RIP_API_SET_ERR_PREFIX_INVALID when the prefix length is not in the range <0–128>

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_aggregate_delete

This function deletes aggregate advertising routes.

This function is called by the following command:

```
no aggregate-address
```

Syntax

```
int
ripng_aggregate_delete (u_int32_t vr_id, int rip_instance,
struct pal_in6_addr *addr, int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`rip_instance`	The instance number
`addr`	The aggregated route prefix.
`plen`	The prefix length for the static RIP route.

Output Parameters

None

Return Values

- RIP_API_SET_ERR_INVALID_VALUE when the address is invalid
- RIP_API_SET_ERR_NETWORK_NOT_EXIST when the network does not exist
- RIP_API_SET_ERR_PREFIX_INVALID when the prefix length is not in the range <0–128>
- RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found
- RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist
- RIP_API_SET_SUCCESS when the call is successful

ripng_cisco_metric_behavior_set

This function sets the metric update as Cisco; updating the metric consistent with Cisco is enabled.

This function is called by the following command:

Syntax

```
int
ripng_cisco_metric_behavior_set (u_int32_t vr_id, int instance,
u_char metric_type);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`metric_type`	The metric type: Cisco behavior or RFC behavior

Output Parameters

None

Return Values

RIP_API_SET_ERR_METRIC_TYPE_INVALID when `metric_type` is greater than RIPNG_CISCO_METRIC_COMPLIANT

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful.Aripng_cisco_metric_behavior_unset

ripng_cisco_metric_behavior_unset

This function unsets updating the metric consistent with Cisco and reverts to the default metric type: RIPng RFC compliant.This function is called by the following command:

```
no
```

Syntax

```
int
ripng_cisco_metric_behavior_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_default_metric_set

This function sets the routing protocol to use the specified metric value for all redistributed routes. The specified default metric will be used by all routes that are redistributed.

This function is called by the following command:

```
default-metric
```

Syntax

```
int
ripng_default_metric_set (u_int32_t vr_id, int instance, int metric);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`metric`	The default metric <0–16>

Output Parameters

None

Return Values

RIP_API_SET_ERR_METRIC_INVALID when the metric is not in range: 0 to RIPNG_METRIC_INFINITY

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_default_metric_unset

This function resets the metrics of the redistributed routes to the default value: 1.

This function is called by the following command:

```
no default-metric
```

Syntax

```
int
ripng_default_metric_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_distance_set_default

This function sets the administrative distance to the specified value. The default value for administrative distance is 120.

This function is called by the following command:

```
distance <1-255>
```

Syntax

```
int
ripng_distance_set_default (u_int32_t vr_id, int instance, char *distance_str)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`distance_str`	The distance value <1-255>

Output Parameters

None

Return Values

RIP_API_SET_ERR_DISTANCE_INVALID when the distance is invalid, not in range <1–255>

RIP_API_SET_ERR_INVALID_VALUE when either the neighbor_prefix or ifname is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_distance_unset_default

This function resets the administrative distance to its default value:120

This function is called by the following command:

```
no distance <1-255>
```

Syntax

```
int
ripng_distance_unset_default (u_int32_t vr_id, int instance)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_enable_nbr_add

This function enables RIP routing on the specified neighbor. This call is used for each connected point-to-point link.

- This function exchanges non-broadcast routing information. It can be used multiple times for additional neighbors.

This function is called by the following command:

```
neighbor
```

Syntax

```
int
ripng_enable_nbr_add (u_int32_t vr_id, int instance, struct pal_in6_addr *addr,
char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`addr`	The neighbor address
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_ADDRESS_INVALID when the neighbor address is invalid

RIP_API_SET_ERR_INVALID_VALUE when either the neighbor_prefix or ifname is NULL

RIP_API_SET_ERR_NBR_STATIC_EXIST when the neighbor is already in the table

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_enable_nbr_delete

This function disables RIPng routing on the neighbor.

This function is called by the following command:

```
no neighbor
```

Syntax

```
int
ripng_enable_nbr_delete (u_int32_t vr_id, int instance,
struct pal_in6_addr *addr, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`addr`	The neighbor address
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_ADDRESS_INVALID when the neighbor address is invalid

RIP_API_SET_ERR_INVALID_VALUE when the neighbor_prefix is NULL

RIP_API_SET_ERR_NBR_STATIC_EXIST when the neighbor is already in the table

RIP_API_SET_ERR_NBR_STATIC_NOT_EXIST when the static node does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_if_ipv6_router_set

This function enables RIPng routing on the interface.

This function is called by the following command:

```
ipv6 router rip
```

Syntax

```
int
ripng_if_ipv6_router_set (u_int32_t vr_id, char *ifname, char *tag);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

`tag`	The RIPng tag

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_INSTANCE_EXIST when the different instance is already configured

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_if_ipv6_router_unset

This function disables RIPng routing on the interface.

This function is called by the following command:

```
no ipv6 router rip
```

Syntax

```
int
ripng_if_ipv6_router_unset (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the instance is not configured

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_if_split_horizon_poisoned_set

This function enables RIP split-horizon poisoned reverse behavior.

- Using the poisoned parameter with this command includes such routes in updates, but sets their metrics to infinity: advertising that these routes are not reachable.
- The default configuration is split-horizon poisoned.

This function is called by the following command:

```
ipv6 rip split-horizon poisoned
```

Syntax

```
int
```

```
ripng_if_split_horizon_poisoned_set (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the instance is not configured

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_if_split_horizon_set

This function enables the RIP split-horizon behavior. This command helps avoid including routes in updates sent to the same gateway from which they were learned.

- Using the split horizon command omits routes learned from one neighbor, in updates sent to that neighbor.
- The default configuration is split-horizon poisoned.

This function is called by the following command:

```
ipv6 rip split-horizon
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	Pointer to the interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

RIP_API_SET_ERR_SPLIT_HORIZON_INVALID when the split horizon is invalid

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_if_split_horizon_unset

This function disables the split horizon behavior. The default configuration is split-horizon poisoned.

This function is called by the following CLI commands:

```
no ip rip split-horizon
ip rip split-horizon poisoned
```

Syntax

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

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_instance_set

This function establishes an instance of the ipV6 RIP router; the RIP routing process is enabled for IPv6.

This function is called by the following command:

```
router ipv6 rip
```

Syntax

```
int
ripng_instance_set (u_int32_t vr_id, int instance)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the instance is not found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_instance_unset

This function removes an instance of the IPv6 router: disables the RIP routing process.

This function is called by the following command:

```
no router ipv6 rip
```

Syntax

```
int
ripng_instance_unset (u_int32_t vr_id, int instance)
```

Input Parameters

`vr_id`	The Virtual Router ID; its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST the instance is not found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_offset_list_set

This function adds an offset to in and out metrics to routes learned through RIP: specifies the offset value that is added to the routing metric.

- When the networks match the access list, the offset is applied to the metrics.
- No change occurs when the offset value is zero.

This function is called by the following command:

```
offset-list
```

Syntax

```
int
ripng_offset_list_set (u_int32_t vr_id, int instance, char *alist,
char *direct_str, int metric, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`alist`	The access list name

`direct_str`	The packet direction
`metric`	The metric or offset
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when access_list_str, direction or metric is NULL

RIP_API_SET_ERR_METRIC_INVALID when the metric is not in range: 0 to RIPNG_METRIC_INFINITY

RIP_API_SET_ERR_OFFSET_LIST_NOT_EXIST when the OFFSET LIST does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_offset_list_unset

This function removes the offset list: the offset value is removed from the routing metric.

This function is called by the following command:

```
no offset-list
```

Syntax

```
int
ripng_offset_list_unset (u_int32_t vr_id,int instance, char *alist,
char *direct_str, int metric, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`alist`	The access list name
`direct_str`	The packet direction
`int metric`	The metric or offset
`char *ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when `access_list_str`, `direction` or `metric` is NULL

RIP_API_SET_ERR_METRIC_INVALID when the `metric` is not in range: 0 to RIPNG_METRIC_INFINITY

RIP_API_SET_ERR_OFFSET_NOT_EXIST when the OFFSET LIST does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_passive_if_add

This function suppresses RIP updates: blocks RIP broadcast on the interface.

This function is called by the following command:

```
passive-interface
```

Syntax

```
int
ripng_passive_if_add (u_int32_t vr_id,int instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_EXIST when the interface already exists

RIP_API_SET_ERR_INVALID_VALUE when `ifname` is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_passive_if_delete

This function disables blocking RIP broadcasts on the interface.

This function is called by the following command:

```
no passive-interface
```

Syntax

```
int
ripng_passive_if_delete (u_int32_t vr_id,int instance, char *ifname);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance T`	he instance number

`ifname`	The interface name

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_EXIST when the interface already exists

RIP_API_SET_ERR_INVALID_VALUE when ifname is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_recvbuf_size_set

This function sets the specified size of the RIP UDP buffer.

This function is called by the following command:

```
recv-buffer-size
```

Syntax

```
int
ripng_recvbuf_size_set (u_int32_t vr_id, int instance, u_int32_t bufsize);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`bufsize`	The size of the receiving buffer

Output Parameters

None

Return Values

RIP_API_SET_ERR_CANT_CHANGE_BUFFER_SIZE when the buffer size cannot be set by the lower layer, such as kernel

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_recvbuf_size_unset

This function resets the size of the RIP UDP buffer to the default value: (1024*192).

This function is called by the following command:

```
no recv-buffer-size
```

Syntax

```
int
ripng_recvbuf_size_set (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance

Output Parameters

None

Return Valuest

RIP_API_SET_ERR_CANT_CHANGE_BUFFER_SIZE when the buffer size cannot be set by the lower layer, such as kernel

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_redistribute_metric_set

This function implements the redistribute metric route map command to specify the metric of the route map; redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIPng.

- Kernel, connected and static are distributed into the RIPng.
- It uses the route map configured and sets the metric for redistribution of routes.

This function is called by the following command:

```
redistribute metric
```

Syntax

```
int
ripng_redistribute_metric_set (u_int32_t vr_id,int instance, char type_str,
int metric);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`instance`	The instance number
`type_str`	The type of route: kernel, connected, static, OSPF, BGP
`metric`	The metric value <0–16>

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when route_type is NULL or not one of the following: kernel; connect; static; ospf; bgp

RIP_API_SET_ERR_METRIC_INVALID when the metric is not in range: 0 to RIPNG_METRIC_INFINITY

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_redistribute_metric_rmap_set

This function implements the redistribute metric route map command to specify the metric of the route map; redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIPng.

- Kernel, connected and static are distributed into the RIPng.
- It uses the route map configured and sets the metric for redistribution of routes.

This function is called by the following command:

`redistribute metric`

Syntax

```
int
ripng_redistribute_metric_rmap_set (u_int32_t vr_id, int instance,
char *type_str, int metric, char *name)
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The number of the instance
`type_str`	The type of route: kernel, connected, static, OSPF, BGP
`metric`	The metric value <0–16>
`name`	The route map name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE the `type_str` or name is NULL

RIP_API_SET_ERR_METRIC_INVALID when the metric value out of range: 0 and RIPNG_METRIC_INFINITY

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance is not found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

ripng_redistribute_rmap_set

This function implements the redistribute metric route map command to specify the metric of the route map; redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIPng.

- Kernel, connected and static are distributed into the RIPng.
- It uses the route map configured and sets the metric for redistribution of routes.

This function redistributes information from other routing protocols.

This function is called by the following command:

```
redistribute route-map
```

Syntax

```
int
ripng_redistribute_rmap_set (u_int32_t vr_id,int instance, char *type_str,
char *name);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`type_str`	The type of route: kernel, connected, static, OSPF, BGP
`name`	The route map name

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when either route_type or route_map_str is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_ROUTE_PROTO_INVALID when not one of these: kernel; connect; static; ospf; bgp

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_redistribute_set

This function implements the redistribute metric route map command to specify the metric of the route map; redistributes routes learned from other routing protocols (OSPF, IS-IS, BGP) to RIPng.

- Kernel, connected and static are distributed into the RIPng.

This function is called by the following command:

```
redistribute
```

Syntax

```
int
ripng_redistribute_set (u_int32_t vr_id,int instance, char *type_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`type_str`	The type of route: kernel, connected, static, OSPF, BGP

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when route_type is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_redistribute_unset

This function disables the redistribution of routes into RIPng, and removes the configured route map and metric.

This function is called by the following command:

```
no redistribute
```

Syntax

```
int
ripng_redistribute_unset (u_int32_t vr_id, int instance, char *type_str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`type_str`	The type of route: kernel, connected, static, OSPF, BGP

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when route_type is NULL

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_route_add

This function configures a static route for advertisement through RIP explicitly. An ideal configuration includes a static route that is redistribute via redistribute static inside a routing process.This command eliminates that overhead; efficient for debug.

This function is called by the following command:

```
route
```

Syntax

```
int
ripng_route_add (u_int32_t vr_id,int instance, struct pal_in6_addr *addr,
int plen);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`addr`	The address of the source prefix
`plen`	The prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when ifname is NULL

RIP_API_SET_ERR_NETWORK_EXIST when the network already exists

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_route_default_add

This function generates a default route into the Routing Information Protocol (RIP).

This function is called by the following command:

```
default-information originate
```

Syntax

```
int
ripng_route_default_add (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number

Output Parameters

None

Return Values

RIP_API_SET_ERR_NETWORK_EXIST when the network already exists

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_route_default_delete

This function disables the configuration of the default route into the Routing Information Protocol (RIP).

This function is called by the following command:

```
no default-information originate
```

Syntax

```
int
ripng_route_default_delete (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number

Output Parameters

None

Return Values

RIP_API_SET_ERR_NETWORK_NOT_EXIST when the network does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_route_delete

This function unsets the configured RIPng static route.

This function is called by the following command:

```
no route
```

Syntax

```
int
ripng_route_delete (u_int32_t vr_id, int instance, struct pal_in6_addr *addr);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`addr`	The address of the source prefix
`plen`	The prefix length for the static RIP route

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when ifname is NULL

RIP_API_SET_ERR_NETWORK_NOT_EXIST when the network does not exist

RIP_API_SET_ERR_PREFIX_INVALID when the source address is invalid

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_route_type_delete

This function clears specified data from the RIPng routing table.

This function is called by the following command:

```
clear ipv6 rip route
```

Syntax

```
int
ripng_route_type_delete (u_int32_t vr_id, int instance, char *str);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter
`instance`	The instance number
`str`	The route type: `kernel`, `static`, `connected`, `rip`, `ospf`, `isis`, `bgp`, or `all`

Output Parameters

None

Return Values

RIP_API_SET_ERR_INVALID_VALUE when the router type is NULL

RIP_API_SET_ERR_NETWORK_NOT_EXIST when the network does not exist

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_timers_set

This function sets the specified time per RIP timer: update timer; timeout timer; garbage timer.

- At the specified interval, the update timer sends an update containing the complete routing table to every neighboring router. When the time specified by the timeout parameter expires the route is no longer valid.
- For a short period, the routing information is retained in the routing table so that neighbors are notified that the route has been dropped.The route is included in all updates until the specified garbage time expires.
- All routers in the network must have the same timers to allow RIP to execute a distributed and asynchronous routing algorithms. The timers should not be synchronized as it might lead to unnecessary collisions on the network.

This function is called by the following command:

```
timer
```

Syntax

```
int
rip_timers_set (u_int32_t vr_id, int instance, u_int32_t update,
u_int32_t timeout, u_int32_t garbage);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number
`update`	The number of update timer seconds; default value is 30
`timeout`	The number of timeout timer seconds; default value is 180
`garbage`	The number of garbage timer seconds; default value is 120

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

ripng_timers_unset

This call resets the three timers to the default values:

- Update timer, 30 seconds
- Timeout timer, 180 seconds

- Garbage timer, 120 seconds

This function is called by the following command:

```
no timer
```

Syntax

```
int
ripng_timers_unset (u_int32_t vr_id, int instance);
```

Input Parameters

`vr_id`	The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter.
`instance`	The instance number

Output Parameters

None

Return Values

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

RIP_API_SET_SUCCESS when the call is successful

CHAPTER 5 SNMP API

The chapter contains the RIP SNMP tables and API.

Note: The SNMP functions only apply to RIP; there are no SNMP functions for RIPng.

MIB Overview

The RIP-2 MIB contains global counters, which can be used for detecting RIP incompatibilities via interface-specific statistics, configuration information, and relationships. This information is stored in Interface and Peer tables.

The `ripv2-mib.txt` file contains the MIB definitions for SNMP (based on the definitions in RFC 1724). This file is divided into three sections:

- Global Counters—RIP-2 counters group
- Interface Table—RIP-2 interfaces groups
- Peer Table—RIP-2 peer group

Global Counters

rip2GlobalRouteChanges

Attribute	Syntax	Access	Function
rip2GlobalRouteChanges	Counter32	read-only	rip2_get_global_route_changes
rip2GlobalQueries	Counter32	read-only	rip2_get_global_queries

Interface Table

rip2IfStatEntry

Attribute	Syntax	Access	Function
rip2IfStatAddress	IpAddress	read-only	rip2_get_if_stat_addr rip2_get_next_if_stat_addr
rip2IfStatRcvBadPackets	Counter32	read-only	rip2_get_if_stat_rcv_bad_packets rip2_get_if_stat_rcv_bad_packets
rip2IfStatRcvBadRoutes	Counter32	read-only	rip2_get_if_stat_rcv_bad_routes rip2_get_next_if_stat_rcv_bad_routes
rip2IfStatSentUpdates	Counter32	read-only	rip2_get_if_stat_sent_updates rip2_get_next_if_stat_sent_updates
rip2IfStatStatus	RowStatus	read-create	rip2_get_if_stat_status rip2_get_next_if_stat_status rip2_set_if_stat_status

Attribute	Syntax	Access	Function
rip2IfConfAddress	IpAddress	read-only	rip2_get_if_conf_address rip2_get_next_if_conf_address
rip2IfConfDomain	RouteTag	read-create	rip2_get_if_conf_domain rip2_get_next_if_conf_domain rip2_set_if_conf_domain
rip2IfConfAuthType	INTEGER	read-create	rip2_get_if_conf_auth_type rip2_get_next_if_conf_auth_type rip2_set_if_conf_auth_type
rip2IfConfAuthKey	OCTET STRING	read-create	rip2_get_if_conf_auth_key rip2_get_next_if_conf_auth_key rip2_set_if_conf_auth_key
rip2IfConfSend	INTEGER	read-create	rip2_get_if_conf_send rip2_get_next_if_conf_send rip2_set_if_conf_send
rip2IfConfReceive	INTEGER	read-create	rip2_get_if_conf_receive rip2_get_next_if_conf_receive rip2_set_if_conf_receive
rip2IfConfDefaultMetric	INTEGER	read-create	rip2_get_if_conf_default_metric rip2_get_next_if_conf_default_metric rip2_get_next_if_conf_default_metric
rip2IfConfStatus	RowStatus	read-create	rip2_get_if_conf_status rip2_get_next_if_conf_status rip2_set_if_conf_status
rip2IfConfSrcAddress	IpAddress	read-create	rip2_get_if_conf_src_address rip2_get_next_if_conf_src_address rip2_set_if_conf_src_address

Peer Table

rip2PeerTable

Attribute	Syntax	Access	Function
rip2PeerAddress	IpAddress	read-only	rip2_get_peer_address rip2_get_next_peer_address
rip2PeerDomain	RouteTag	read-only	rip2_get_peer_domain rip2_get_next_peer_domain
rip2PeerLastUpdate	TimeTicks	read-only	rip2_get_peer_last_update rip2_get_next_peer_last_update
rip2PeerVersion	INTEGER	read-only	rip2_get_peer_version rip2_get_next_peer_version

Attribute	Syntax	Access	Function
rip2PeerRcvBadPackets	Counter32	read-only	rip2_get_peer_rcv_bad_packets rip2_get_next_peer_rcv_bad_packets
rip2PeerRcvBadRoutes	Counter32	read-only	rip2_get_peer_rcv_bad_routes rip2_get_next_peer_rcv_bad_routes

API

rip2_get_global_route_changes

This function returns the number of route changes made to the IP Route Database by RIP.

Syntax

```
int
rip2_get_global_route_changes (int pid, int *value);
```

Input Parameters

`pid` The RIP Process ID

Output Parameters

`value` The number of route changes

Return Values

RIP_API_GET_ERROR when the call does not succeed

RIP_API_GET_SUCCESS when the call returns the number of route changes obtained

rip2_get_global_queries

This function returns the number of responses sent to RIP queries from other systems.

Syntax

```
int
rip2_get_global_queries (int pid, int *value);
```

Input Parameters

`pid` The RIP Process ID

Output Parameters

`value` The number of responses

Return Values

RIP_API_GET_SUCCESS when the call returns the number of responses obtained

RIP_API_GET_ERROR when the call does not return the number of responses

rip2_get_if_stat_addr

This function returns the IP address of this system on the indicated subnet.

Syntax

```
int
rip2_get_if_stat_addr(int pid, struct pal_in4_addr *addr,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`outaddr`	The IP address

Return Values

RIP_API_GET_ERROR when the does not call find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_next_if_stat_addr

This function returns the IP address of this system on the next indicated subnet.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_stat_addr(int pid, struct pal_in4_addr *addr, int indexlen,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`outaddr`	The IP address

Return Values

RIP_API_GET_ERROR when the call does not find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_if_stat_rcv_bad_packets

This function returns the number of RIP packets received by the RIP process that were discarded for any reason, such as an unknown command type.

Syntax

```
int
rip2_get_if_stat_rcv_bad_packets(int pid, struct pal_in4_addr *addr,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The number of discarded RIP packets

Return Values

RIP_API_GET_ERROR when the call does not succeed

RIP_API_GET_SUCCESS when the call finds the number of discarded RIP packets

rip2_get_next_if_stat_rcv_bad_routes

This function returns the number of routes in valid RIP packets, that were ignored for any reason (e.g. unknown address family) on the next interface

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned
- If indexlen is zero, the first entry is returned

Syntax

```
int
rip2_get_next_if_stat_rcv_bad_routes(int pid, struct pal_in4_addr *addr,
int indexlen, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The number of routes ignored

Return Values

RIP_API_GET_ERROR when the call does not find the number of ignored routes

RIP_API_GET_SUCCESS when the call finds the number of ignored routes

rip2_get_if_stat_rcv_bad_routes

This function gets the next return value for the next interface.

This function returns the number of routes in valid RIP packets that were ignored for any reason (e.g., unknown address family).

Syntax

```
int
rip2_get_if_stat_rcv_bad_routes(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The number of routes ignored

Return Values

RIP_API_GET_ERROR when the call does not find the number of ignored routes

RIP_API_GET_SUCCESS when the call finds the number of ignored routes

rip2_get_next_if_stat_rcv_bad_routes

This function returns the number of routes in valid RIP packets that were ignored for any reason (e.g. unknown address family) on the next interface.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_stat_rcv_bad_routes(int pid, struct pal_in4_addr *addr, int indexlen,
int *value)
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The number of routes ignored

Return Values

RIP_API_GET_ERROR when the call does not succeed

RIP_API_GET_SUCCESS when the call finds the number of ignored routes

rip2_get_if_stat_sent_updates

This function returns the number of triggered RIP updates actually sent on this interface. This does not include full updates containing new information.

Syntax

```
int
rip2_get_if_stat_sent_updates(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The number of triggered RIP updates

Return Values

RIP_API_GET_ERROR when the call does not find the number of triggered RIP updates

RIP_API_GET_SUCCESS when the call finds the number of triggered RIP updates

rip2_get_next_if_stat_sent_updates

This function returns the number of triggered RIP updates actually sent on the next interface. This does not include full updates containing new information.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_stat_sent_updates(int pid, struct pal_in4_addr *addr,
int indexlen, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The number of triggered RIP updates

Return Values

RIP_API_GET_ERROR when the call does not find the number of triggered RIP updates

RIP_API_GET_SUCCESS when the call finds the number of triggered RIP updates

rip2_get_if_stat_status

This function returns the status of the interface.

Syntax

```
int
rip2_get_if_stat_status(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The returned status; RIP_API_STATUS_VALID

Return Values

RIP_API_GET_ERROR when the status is not found

RIP_API_GET_SUCCESS when the status is found

rip2_get_next_if_stat_status

This function returns the status of the next interface.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_stat_status(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	Status; always RIP_API_STATUS_VALID

Return Values

RIP_API_GET_ERROR when the status of the next interface is not found

RIP_API_GET_SUCCESS when the call finds the status of the next interface

rip2_set_if_stat_status

This function sets the status of the specified interface.

Syntax

```
int
rip2_set_if_stat_status (int pid, struct pal_in4_addr *addr, int status);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`status`	The status of the interface

Output Parameters

None

Return Values

RIP_API_SET_ERROR when the status is of unknown value

RIP_API_SET_ERR_IF_NOT_EXIST when the interface of the specified address does not exist

RIP_API_SET_ERR_INCONSISTENT_VALUE when the status is ROW_STATUS_CREATEANDWAIT

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successful

rip2_get_if_conf_address

This function returns the IP address of this system on the indicated subnet.

Syntax

```
int
rip2_get_if_conf_address(int pid, struct pal_in4_addr *addr,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`outaddr`	The IP address of the system on the indicated subnet

Return Values

RIP_API_GET_ERROR when the IP address is not found

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_next_if_conf_address

This function returns the IP address of the next system on the indicated subnet

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned
- If indexlen is zero, the first entry is returned

Syntax

```
int
rip2_get_next_if_conf_address(int pid, struct pal_in4_addr *addr, int indexlen,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`outaddr`	The IP address of the next system on the indicated subnet

Return Values

RIP_API_GET_ERROR when the IP address is not found

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_if_conf_domain

This function returns the value inserted into the Routing Domain field of all RIP packets sent on this interface.

Syntax

```
int
rip2_get_if_conf_domain(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The Routing Domain field value; zero is returned

Return Values

RIP_API_GET_SUCCESS when the call finds the Routing Domain field value

RIP_API_GET_ERROR when the Routing Domain field value is not found

rip2_get_next_if_conf_domain

This function returns the value inserted into the Routing Domain field of all RIP packets sent on the next interface.

- If the IP Address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_domain(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The Routing Domain field value; zero is returned

Return Values

RIP_API_GET_ERROR when no route information is found; no variable is obtained

RIP_API_SUCCESS when the call finds the Routing Domain field value

rip2_set_if_conf_domain

This function sets the value inserted into the Routing Domain field of all RIP packets sent on this interface.

Syntax

```
int
rip2_set_if_conf_domain (int pid, struct pal_in4_addr *addr, int intval);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`intval`	The Routing Domain field value

Output Parameters

None

Return Values

RIP_API_SET_ERROR when the set is obsolete and not supported

rip2_get_if_conf_auth_type

This function returns the type of authentication used on this interface.

Syntax

```
int
rip2_get_if_conf_auth_type (int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The authentication type:
`1`	No authentication
`2`	Simple password
`3`	MD5 authentication

Return Values

RIP_API_GET_ERROR when the authentication type is not found

RIP_API_GET_SUCCESS when the call finds the authentication type

rip2_get_next_if_conf_auth_type

This function returns the type of authentication used on the next interface.

- If the IP Address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- if indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_auth_type(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The authentication type
`value`	The authentication type:
`1`	No authentication
`2`	Simple password
`3`	MD5 authentication

Return Values

RIP_API_GET_ERROR when the authentication type is not found

RIP_API_GET_SUCCESS when the call finds the authentication type

rip2_set_if_conf_auth_type

This function sets the authentication type used on this interface.

Syntax

```
int
rip2_set_if_conf_auth_type (int pid, struct pal_in4_addr *addr, int intval);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`intval`	The authentication type

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERROR when `intval` is unknown

rip2_get_if_conf_auth_key

This function returns the value to be used as the authentication key whenever the corresponding instance of rip2_get_if_conf_auth_type has a value other than noAuthentication.

Syntax

```
int
rip2_get_if_conf_auth_key(int pid, struct pal_in4_addr *addr, char **key);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`key`	The authentication key value

Return Values

RIP_API_GET_ERROR when the call does not find the authentication key value

RIP_API_GET_SUCCESS when the call finds the authentication key value

rip2_get_next_if_conf_auth_key

This function returns the value to be used as the next authentication key whenever the corresponding instance of rip2_get_if_conf_auth_type has a value other than noAuthentication.

- If the IP Address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_auth_key(int pid, struct pal_in4_addr *addr, int indexlen,
char **key);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use
`key`	The authentication key value

Output Parameters

`value`	The next authentication key value

Return Values

RIP_API_GET_ERROR when the next authentication key value is not found

RIP_API_GET_SUCCESS when the call finds the next authentication key value

rip2_set_if_conf_auth_key

This function sets the value to be used as the authentication key of the corresponding instance of RIP.

Syntax

```
int
rip2_set_if_conf_auth_key (int pid, struct pal_in4_addr *addr, char *key);
```

Input Parameters

`pid`	The RIP Process ID

`addr`	The IP address of a specified interface
`key`	The authentication key value

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when the interface of the specified address is not present

RIP_API_SET_SUCCESS when the auth string is successfully set

rip2_get_if_conf_send

This function returns what the router sends on this interface (typically updates).

Syntax

```
int
rip2_get_if_conf_send(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The updates

Return Values

RIP_API_GET_ERROR when the updates sent by the router are not found

RIP_API_GET_SUCCESS when the call finds the updates sent by the router

rip2_get_next_if_conf_send

This function returns what the router sends on the next interface (typically updates).

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_send(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID

`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	One of the following updates: RIP_API_IFCONF_SEND_DONOTSEND RIP_API_IFCONF_SEND_RIPVERSION1 RIP_API_IFCONF_SEND_RIPVERSION2 RIP_API_IFCONF_SEND_RIP1COMPATIBLE

Return Values

RIP_API_GET_ERROR when the next updates sent by the router are not found

RIP_API_GET_SUCCESS when the call finds the next updates sent by the router

rip2_set_if_conf_send

This function sets the RIP version to be sent in the control packet.

Syntax

```
int
rip2_set_if_conf_send (int pid, struct pal_in4_addr *addr, int intval);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`intval`	The updates

Output Parameters

None

Return Values

RIP_API_SET_ERR_IF_NOT_EXIST when the interface of the specified address is not present

RIP_API_SET_ERR_INVALID_VALUE when the `intval` is of unknown value

rip2_get_if_conf_receive

This function returns the version of RIP updates that is to be accepted. The return value: 1 for rip1; 2 for rip2; 3 for either rip1 or rip2; 4 if none of the versions are to be accepted. The default value is 3 for either rip1 or rip2. Rip2 and rip1 or rip2 imply reception of multicast packets.

Syntax

```
int
rip2_get_if_conf_receive(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The version of RIP updates

Return Values

RIP_API_GET_ERROR when the version is not accepted

RIP_API_GET_SUCCESS when the call finds the version to be accepted

rip2_get_next_if_conf_receive

This function returns the version of RIP updates received by the next interface.

- If the IP Address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_if_conf_receive(int pid, struct pal_in4_addr *addr, int indexlen, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The version of RIP updates. The returned values can be one of the following: RIP_API_IFCONF_RECV_DONOTRECEIVE RIP_API_IFCONF_RECV_RIP1 RIP_API_IFCONF_RECV_RIP2 RIP_API_IFCONF_RECV_RIP1ORRIP2

Return Values

RIP_API_GET_ERROR when the version to be accepted is not found

RIP_API_GET_SUCCESS when the call finds the version to be accepted

rip2_set_if_conf_receive

This function sets the version of RIP updates to be accepted.

- The return value is 1 for rip1, 2 for rip2, 3 for either rip1 or rip2, and 4 if none of the versions are to be accepted.

- The default value is 3 for either rip1 or rip2.
- Rip2 and rip1 or rip2 imply reception of multicast packets.

Syntax

```
int
rip2_set_if_conf_receive (int pid, struct pal_in4_addr *addr, int intval);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`intval`	The updates

Output Parameters

None

Return Values

RIP_API_SET_ERROR if the update is of any other type

RIP_API_SET_ERR_IF_NOT_EXIST when the interface of the specified address is not present

rip2_get_if_conf_default_metric

This function returns the variable that indicates the metric that is to be used for the default route entry in RIP updates originated on this interface. A value of zero (0) indicates that no default route should be originated; in such a case, a default route via another router may be propagated.

Syntax

```
int
rip2_get_if_conf_default_metric(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The default metric

Return Values

RIP_API_GET_ERROR when no route information is found; no variable is obtained

RIP_API_GET_SUCCESS when the variable is obtained

rip2_get_next_if_conf_default_metric

This function returns the variable that indicates the metric that is to be used for the default route entry in RIP updates originated on the next interface. A value of zero (0) indicates that no default route should be originated; in such a case, a default route via another router may be propagated.

- If the IP Address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- if indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_default_metric(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The default metric value

Return Values

RIP_API_GET_ERROR when no route information is found; no variable is obtained

RIP_API_GET_SUCCESS when the route information is found

rip2_set_if_conf_default_metric

This function sets the variable that indicates the metric to be used for the default route entry in RIP updates originated on this interface. A value of zero (0) indicates that no default route should be originated; in such a case, a default route via another router may be propagated.

Syntax

```
int
rip2_set_if_conf_default_metric (int pid, struct pal_in4_addr *addr, int intval);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`intval`	The default metric

Output Parameters

None

Return Values

RIP_API_SET_ERROR because “set default route metric per interface” is not supported

rip2_get_if_conf_status

This function returns the status of the interface.

Syntax

```
int
rip2_get_if_conf_status(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`value`	The status of the interface.
	RIP_API_STATUS_VALID when RIP protocol is enabled on the interface
	RIP_API_STATUS_INVALID when RIP protocol is disabled on the interface

Return Values

RIP_API_GET_SUCCESS when the call finds the status

RIP_API_GET_ERROR when the status in not available

rip2_get_next_if_conf_status

This function returns the status of the next interface.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_status(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`value`	The status: currently, RIP_API_STATUS_VALID is returned

Return Values

RIP_API_GET_ERROR when the call does not find the status of the next interface

RIP_API_GET_SUCCESS if the call finds the status of the next interface

rip2_set_if_conf_status

This function sets the status of the interface.

Syntax

```
int
rip2_set_if_conf_status (int pid, struct pal_in4_addr *addr, int status);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`status`	The interface status

Output Parameters

None

Return Values

RIP_API_SET_ERROR when the status value is unknown

RIP_API_SET_ERR_IF_NOT_EXIST when the interface with the specified address is not found

RIP_API_SET_ERR_INCONSISTENT_VALUE when the value of status is ROW_STATUS_CREATEANDWAIT and ROW_STATUS_CREATEANDGO

RIP_API_SET_ERR_INVALID_VALUE when the status value is not within range

RIP_API_SET_ERR_PROCESS_NOT_EXIST when the instance cannot be found

RIP_API_SET_ERR_VR_NOT_EXIST when `vr_id` does not exist

RIP_API_SET_SUCCESS when it is successfully set

rip2_get_if_conf_src_address

This function returns the IP address the system will use as a source address on this interface.

Syntax

```
int
rip2_get_if_conf_src_address(int pid, struct pal_in4_addr *addr,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface

Output Parameters

`outaddr`	The IP address that will be used as a source address

Return Values

RIP_API_GET_ERROR when the call does not find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_next_if_conf_src_address

This function returns the IP address the system will use as a source address on the next interface.

- If the IP address or interface index are partial, and a match is located in the table, that entry is returned. If no match, then error is returned.
- If the IP address or interface index are complete, and a match is located, the next entry is returned. If no next entry, then error is returned.
- If indexlen is zero, the first entry is returned.

Syntax

```
int
rip2_get_next_if_conf_src_address(int pid, struct pal_in4_addr *addr,
int indexlen, struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The IP address of a specified interface
`indexlen`	Reserved for future use

Output Parameters

`outaddr`	The IP address that will be used as a source address

Return Values

RIP_API_GET_ERROR when the call does not find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_set_if_conf_src_address

This function sets the IP address the system will use as a source address on this interface.

Syntax

```
int
rip2_set_if_conf_src_address (int pid, struct pal_in4_addr *new_addr,
struct pal_in4_addr *addr)
```

Input Parameters

`pid`	The RIP Process ID
`new_addr`	The IP address that will be used as a source address
`addr`	The IP address of a specified interface

Output Parameters

None

Return Values

RIP_API_SET_ERROR when the `new_addr` is not successfully established

RIP_API_SET_SUCCESS when the `new_addr` is successfully established

rip2_get_peer_address

This function returns the IP address that the peer is using as its source address.

Syntax

```
int
rip2_get_if_peer_address(int pid, struct pal_in4_addr *addr,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	he peer address

Output Parameters

`outaddr`	The IP address of the peer

Return Values

RIP_API_GET_ERROR when the call does not find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_next_peer_address

This function returns the IP address that the next peer is using as its source address.

Syntax

```
int
rip2_get_next_peer_address(int pid, struct pal_in4_addr *addr, int indexlen,
struct pal_in4_addr *outaddr);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`outaddr`	The IP address of the peer

Return Values

RIP_API_GET_ERROR when the call does not find the IP address

RIP_API_GET_SUCCESS when the call finds the IP address

rip2_get_peer_domain

This function returns the value in the Routing Domain field in RIP packets received from the peer. As domain support is deprecated, this value goes to zero (0).

Syntax

```
int
rip2_get_peer_domain(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address

Output Parameters

`value`	The Routing Domain field value

Return Values

RIP_API_GET_ERROR when the call does not find the Routing Domain value

RIP_API_GET_SUCCESS when the call finds the Routing Domain value

rip2_get_next_peer_domain

This function returns the value in the Routing Domain field in RIP packets received from the next peer. As domain support is deprecated, this value goes to zero (0).

Syntax

```
int
rip2_get_next_peer_domain(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`value`	The Routing Domain field value

Return Values

RIP_API_GET_ERROR when the call does not find the Routing Domain value

RIP_API_GET_SUCCESS when the call finds the Routing Domain value

rip2_get_peer_last_update

This function returns the value of sysUptime when the most recent RIP Update is received from this system.

Syntax

```
int
rip2_get_if_peer_last_update(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address

Output Parameters

`value`	The sys uptime value.

Return Values

RIP_API_GET_ERROR when the call does not find the sysUptime value

RIP_API_GET_SUCCESS when the call finds the sysUptime value

rip2_get_next_peer_last_update

This function returns the value of sysUptime of the next peer when the most recent RIP Update is received from this system.

Syntax

```
int
rip2_get_next_peer_last_update(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`value`	The sysUptime value

Return Values

RIP_API_GET_ERROR when the call does not find the `sysUptime` value

RIP_API_GET_SUCCESS when the call finds the `sysUptime` value

rip2_get_peer_version

This function returns the RIP version number in the header of the last RIP packet received.

Syntax

```
int
rip2_get_peer_version(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address

Output Parameters

`value`	The version number

Return Values

RIP_API_GET_ERROR when the call does not the version number

RIP_API_GET_SUCCESS when the call finds the version number

rip2_get_next_peer_version

This function returns the RIP version number of the next peer in the header of the last RIP packet received.

Syntax

```
int
rip2_get_next_peer_version(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`value`	The version number

Return Values

RIP_API_GET_ERROR when the version number is not found

RIP_API_GET_SUCCESS when the call finds the version number

rip2_get_peer_rcv_bad_packets

This function returns the number of RIP response packets from this peer discarded as invalid.

Syntax

```
int
rip2_get_peer_rcv_bad_packets(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID

`addr`	The peer address

Output Parameters

`value`	The number of discarded RIP response packets

Return Values

RIP_API_GET_ERROR when the call does not find the number of packets discarded

RIP_API_GET_SUCCESS when the call finds the number of packets discarded

rip2_get_next_peer_rcv_bad_packets

This function returns the number of RIP response packets from the next peer discarded as invalid.

Syntax

```
int
rip2_get_next_peer_rcv_bad_packets(int pid, struct pal_in4_addr *addr,
int indexlen, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`value`	The number of discarded RIP response packets

Return Values

RIP_API_GET_ERROR when the call does not find the number of packets discarded

RIP_API_GET_SUCCESS when the call finds the number of packets discarded

rip2_get_peer_rcv_bad_routes

This function returns the number of routes from this peer that were ignored because the entry format was invalid.

Syntax

```
int
rip2_get_peer_rcv_bad_routes(int pid, struct pal_in4_addr *addr, int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address

Output Parameters

`value`	The number of ignored routes

Return Values

RIP_API_GET_ERROR when the call does not find the number of ignored routes

RIP_API_GET_SUCCESS when the call finds the number of ignored routes

rip2_get_next_peer_rcv_bad_routes

This function returns the number of routes from the next peer that were ignored because the entry format was invalid.

Syntax

```
int
rip2_get_next_peer_rcv_bad_routes(int pid, struct pal_in4_addr *addr, int indexlen,
int *value);
```

Input Parameters

`pid`	The RIP Process ID
`addr`	The peer address
`indexlen`	Reserved for future use

Output Parameters

`value`	The number of ignored routes

Return Values

RIP_API_GET_ERROR when the call does not find the number of ignored routes

RIP_API_GET_SUCCESS when the call finds the number of ignored routes

Appendix A Source Files

This appendix provides information about source files.

Note: Only files related to external functions are listed.

ripd

This directory contains the source code for RIP IPv4.

Header File	Implementation File	Description
ripd.h	ripd.c	Functions to initiate, stop, and destroy ripd instances
rip_api.h	rip_api.c	Prototypes, functions and data structures for the command, MIB, and other APIs
rip_cli.h	rip_cli.c	Command API
rip_snmp.h	rip_snmp.c	Functions and data structures for handling MIB requests

ripngd

This directory contains the source code for RIP next generation (IPv6).

Header File	Implementation File	Description
ripngd.h	ripngd.c	Functions to initiate, stop, and destroy ripngd instances
ripng_api.h	ripng_api.c	Prototypes, functions, and data structures for the command, MIB, and other APIs
ripng_cli.h	ripng_cli.c	Command API

Index

S

T