ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Unicast Routing Information Base
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 the unicast Routing Information Base (RIB).

Audience

This guide is intended for developers who write code to customize and extend the ZebOS-XP unicast RIB.

Conventions

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

Table P-1: Conventions

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

Contents

This guide contains this chapter:

- Chapter 1, *Unicast Routing Information Base API*

Related Documents

The following guides are related to this document:

- *Unicast Routing Information Base Command Reference*
- *Installation Guide*

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

Support

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

Comments

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

CHAPTER 1 Unicast Routing Information Base API

In ZebOS-XP, the `ribd` process is responsible for maintaining a central Routing Information Base. This chapter describes the API for the `ribd` process.

Overview

A Routing Information Base (RIB) is a data structure stored in a network device that lists the routes to particular network destinations and metrics (distances) associated with those routes. A RIB contains information about the topology of the network immediately around it. Maintaining a RIB by discovering network topology is the primary purpose of dynamic routing protocols such as BGP, RIP, and OSPF. Static fixed routes are added to a RIB by commands. (A RIB is also called a routing table.)

A Forwarding Information Base (FIB) is used to find the proper interface to which an input interface should forward a packet. In contrast to RIBs, FIBs are optimized for fast lookup of destination addresses. (A FIB is also called a forwarding table.)

How ribd Interacts with Protocols and the Kernel

Protocol modules create their own routes, and communicate this protocol-specific information to `ribd` via Inter-Process Communication (IPC) messaging functions. The `ribd` RIB contains all routing information received from routing peers, for example, destination prefix, nexthop information, and distance.

For every known prefix, `ribd` maintains a route node entry in its RIB. The `ribd` process populates this table upon receiving routes from:

- Protocols such as BGP, OSPF, and RIP
- Static routes configured using commands
- The kernel FIB
- Connected routes derived from interface information

Routing protocols use different metrics to calculate the best path for a destination. The best path is sent to the `ribd` process.

The kernel FIB contains the minimum amount of information required to make a forwarding decision on a particular packet, for example, destination prefix and nexthop information. This abbreviated form of the information in the RIB is transferred from `ribd` to the kernel FIB via Netlink sockets.

Figure 1-1 shows how protocols, `ribd`, and the kernel communicate.

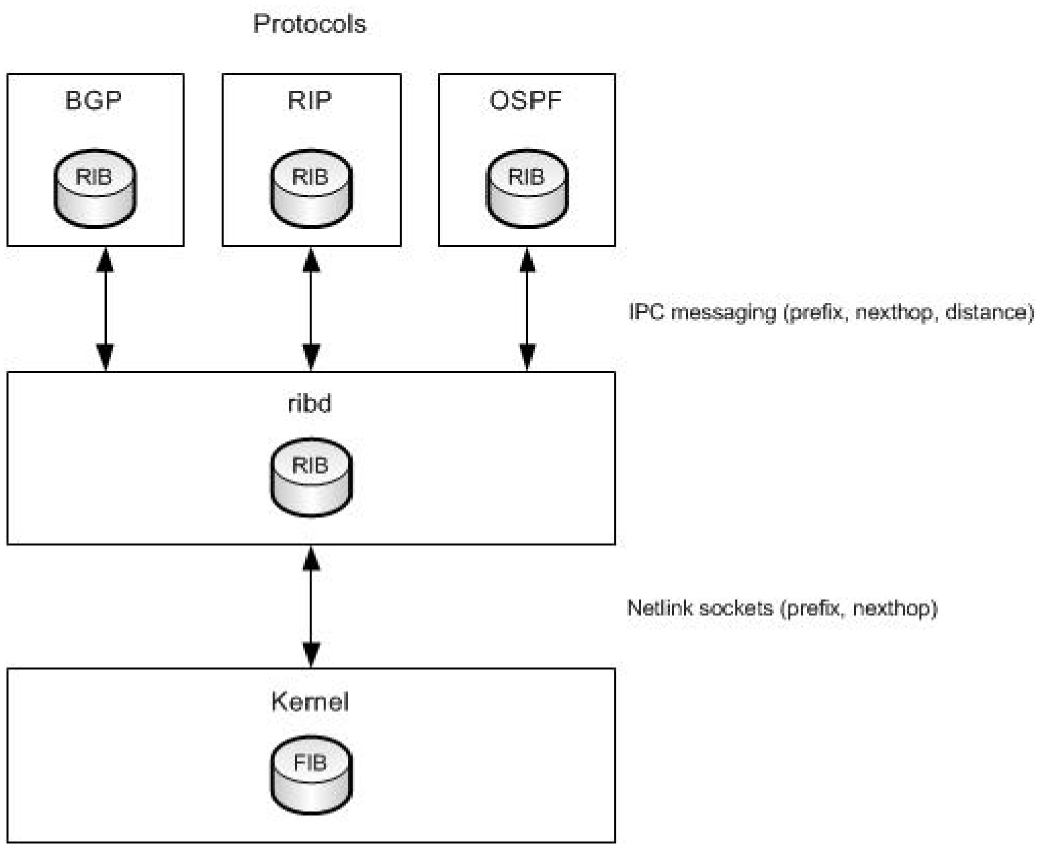
Protocols
BGP
RIB
RIP
RIB
OSPF
RIB
IPC messaging (prefix, nexthop, distance)
ribd
RIB
Netlink sockets (prefix, nexthop)
Kernel
FIB

Figure 1-1: Protocol, ribd, and kernel interaction

With High-Availability or Virtual Routers, there are multiple RIBs:

- Each Virtual Router has a complete copy of the ZebOS-XP RIB and FIB.
- High-Availability systems generally have a primary and backup mode, each containing an RIB, which synchronize with each other.

The following is an example of communication between the RIP protocol RIB and the `ribd` process:

1. RIP receives a packet.
2. RIP processes the data in the packet and determines that it must:
   - Update its RIB table
   - Communicate that information to the `ribd` process
3. The `ribd` process receives the information from RIP and updates the tables in the `ribd` RIB.
4. The `ribd` process sends a message to RIP to acknowledge that the operation has been completed

Selecting Routes to Add to the FIB

This section shows how `ribd` decides the routes to add to the FIB.

When multiple routes are available for the same prefix, `ribd` uses an internal route selection mechanism to select routes to add to the FIB. The primary factor for route selection is the “administrative distance” of the protocol. Table 1-1 lists the default administrative distances of protocols.

Table 1-1: Administrative distance

Protocol	Administrative distance	Preference
Connected	-	1 (highest)
Kernel	-	2
Static	1	3
eBGP	20	4
OSPF	110	5
ISIS	115	6
RIP	120	7
iBGP	200	8 (lowest)

How ribd Adds Routes

The `ribd` process prefers routes learned from protocols with a lower administrative distance over routes learned from protocols with a higher administrative distance.

For example, the following route entries show that the static routes (administrative distance 1) are preferred over the OSPF Route (administrative distance 110):

```
S    *> 10.10.34.0/24 [1/0] via 10.10.31.16, eth2
O     10.10.34.0/24 [110/31] via 10.10.31.16, eth2, 00:21:19
```

Note: The administrative distance of routing protocols can be configured using the `distance` command.

Figure 1-2 shows the steps that `ribd` follows in selecting a route to add to the FIB.

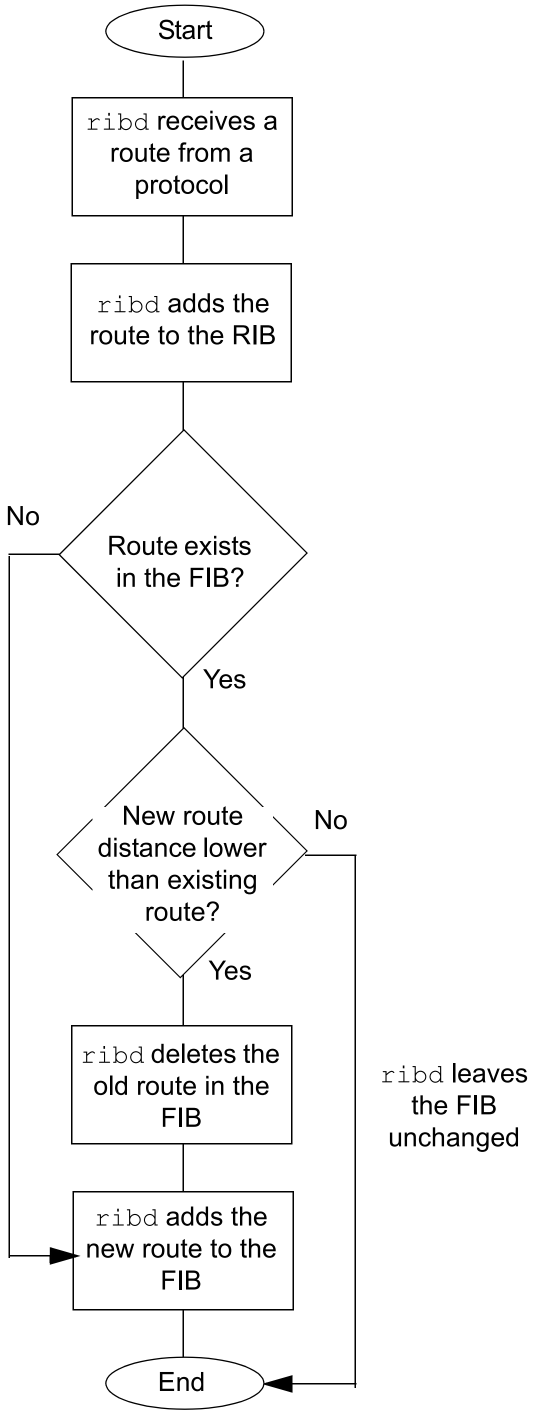
Start
ribd receives a route from a protocol
ribd adds the route to the RIB
Route exists in the FIB?
No
Yes
New route distance lower than existing route?
No
Yes
ribd deletes the old route in the FIB
ribd leaves the FIB unchanged
ribd adds the new route to the FIB
End

Figure 1-2: How ribd adds a route

How ribd Deletes Routes

When `ribd` receives a route delete request from a routing protocol, `ribd` deletes the specified route from its RIB. Then `ribd` checks if the specified route is in the FIB. If the route is in the FIB, `ribd` deletes it from the FIB and checks if another route is in its RIB for the same prefix. If there is another route in the RIB, `ribd` installs this route in the FIB. When multiple such routes exist, `ribd` runs the route selection mechanism to choose the best route to add to the FIB.

Equal-Cost Multipath

The Equal-Cost Multipath (ECMP) feature enables a device to have several next-hops available for a given destination, with each path having the same administrative distance and cost. The `ribd` process can install up to eight ECMP routes.

With ECMP, the kernel can perform load balancing. The algorithm for distributing traffic across ECMP routes is dependent on the kernel, and is usually based on the protocol, source address, destination address and the interface.

Data Structures

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

Common Data Structures

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

- prefix_ipv4
- prefix_ipv6

ipi_vr

See the *Virtual Routing Developer Guide* for an explanation of this struct.

rib_master

This structure in `ribd/ribd.h` stores system-wide settings and variables.

Member name	Description
`vr`	Virtual Router
`zg`	Library-specific globals
`desc`	Description
`module_bits`	Control Virtual Router support per protocol module
`start_time`	RIB master start time
`flags`	Flags
`multipath_num`	Maximum paths
`max_static_routes`	Maximum static routes
`max_fib_routes`	Maximum FIB routes excluding kernel, connected, and static
`fib_retain_time`	Retention time for stale FIB routes when ribd start

Member name	Description
`t_sweep`	Thread: sweep stale FIB routes
`t_rib_kernel_sync`	Thread: RIB kernel synchronization
`t_kernel_msg_stagger`	Thread: hold timer to stagger writes to the kernel
`kernel_msg_stagger_list`	Messages that need to be sent to the kernel
`rmd_conf`	RIB configured debug flag
`rmd_term`	RIB terminal debug flag
`rib_master_cdr_ref`	High availability checkpoint database reference
`rib_master_rib_sweep_tmr_cdr_ref`	High availability checkpoint database reference

Definition

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

  /* RIB pointer to lib_globals */
  struct lib_globals *zg;

  /* Description. */
  char *desc;

  /* Control VR support per PM. */
  modbmap_t module_bits;

  /* RIB master start time.  */
  pal_time_t start_time;

  u_char flags;
#define RIB_MULTIPATH_REFRESH           (1 << 0)
#define RIB_FIB_RETAIN_RESTART          (1 << 1)
#define RIB_IPV4_FORWARDING             (1 << 2)
#define RIB_IPV6_FORWARDING             (1 << 3)

  /* Maximum path config. */
  u_char multipath_num;

  /* Maximum static routes */
  u_int32_t max_static_routes;

  /* Maximum FIB routes excluding Kernel, Connect and Static*/
  u_int32_t max_fib_routes;
  /* The time of retaining stale FIB routes when RIBd start. */
  u_int16_t fib_retain_time;
#define RIB_FIB_RETAIN_TIME_MIN         1
#define RIB_FIB_RETAIN_TIME_MAX         65535
#define RIB_FIB_RETAIN_TIME_DEFAULT     60
#define RIB_FIB_RETAIN_TIME_FOREVER     0

  /* Threads. */
  struct thread *t_sweep;       /* Sweep stale FIB routes. */
#ifdef HAVE_KERNEL_ROUTE_SYNC
  struct thread *t_rib_kernel_sync;     /* RIB kernel sync. */
#endif /* HAVE_KERNEL_ROUTE_SYNC */

#ifdef HAVE_STAGGER_KERNEL_MSGS
  /* Hold timer to stagger writes to the kernel. */
  struct thread *t_kernel_msg_stagger;

  /* List for storing messages that need to be sent to the kernel. */
  struct list *kernel_msg_stagger_list;
#endif /* HAVE_STAGGER_KERNEL_MSGS */

  /* RIB configured and terminal debug flags.  */
  struct
  {
    struct rib_debug_flags rmd_conf;
    struct rib_debug_flags rmd_term;
  } rm_debug;

#ifdef HAVE_HA
  HA_CDR_REF  rib_master_cdr_ref;
  HA_CDR_REF  rib_master_rib_sweep_tmr_cdr_ref;
#endif /* HAVE_HA */
};
```

Command API

The functions in this section are called by the commands in the *Unicast Routing Information Base Command Reference*.

Function	Description
rib_api_multipath_num_func	Sets the maximum number of paths to install in the FIB
rib_cli_ip_route_prefix	Creates a static route
rib_cli_ipv6_route_prefix	Creates a static route
rib_cli_no_ip_route	Deletes all IPv4 static routes for a given prefix in the default VRF instance

Function	Description
rib_cli_no_ip_route_all_vrf	Deletes all IPv4 static routes for a given prefix in all VRF instances
rib_cli_no_ip_route_prefix	Deletes a static route
rib_cli_no_ipv6_route_prefix	Deletes a static route
rib_fib_retain_set	Sets the retention time for stale routes in the FIB during `ribd` restart
rib_fib_retain_unset	Sets the retention time for stale routes in the FIB during `ribd` restart to the default
rib_ip_vrf_isid_set	Creates a VRF instance associated with an I-SID
rib_ip_vrf_unset	Deletes a VRF instance
rib_ipv4_route_set	Creates a static route
rib_ipv4_route_stale_clear	Clears stale IPv4 routes from the RIB and FIB
rib_ipv4_route_unset	Deletes a static route
rib_ipv6_route_set	Creates a static route
rib_ipv6_route_stale_clear	Clears stale IPv6 routes from the RIB and FIB
rib_ipv6_route_unset	Deletes a static route
rib_ipv6_route_unset_all	Deletes all static IPv4 route for a given VRF instance and prefix
rib_ipv6_route_unset_all_vrf	Deletes all static IPv6 routes for a given prefix from all VRF instances
rib_set_maximum_fib_routes	Sets the maximum number of FIB routes excluding kernel, connected, and static routes
rib_set_maximum_static_routes	Sets the maximum number of static routes
rib_unset_maximum_fib_routes	Sets the maximum number of FIB routes excluding kernel, connected, and static routes to the default
rib_unset_maximum_static_routes	Sets the maximum number of static routes to the default

Include File

Except where noted otherwise, you need to include `ribd/rib_api.h` to call the functions in this section.

rib_api_multipath_num_func

The function sets the maximum number of paths to install in the FIB (Forwarding Information Base) for the ECMP (Equal-Cost MultiPath) feature.

This function is called by the `maximum-paths` command.

Syntax

```
int
rib_api_multipath_num_func (int vr_id, int multipath, int set)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0
`multipath`	Maximum number of paths to install in the FIB
`set`	Whether to set the default number of paths; if you specify `PAL_FALSE`, the value of `DEFAULT_MULTIPATH_NUM` in `ribd/ribd.h` is used and `multipath` is ignored

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_ip_route_prefix

The function creates a static route.

This function is called by the `ip route` command.

You can specify:

- Only a nexthop interface name *or* nexthop address in `gate_str`
- Both a nexthop interface name in `ifn` *and* a nexthop address in `gate_str`

Syntax

```
int
rib_cli_ip_route_prefix (struct rib_master *rm,
                         vrf_id_t vrf_id, struct prefix_ipv4 *p,
                         char *ifn, char *gate_str, int distance,
                         u_int32_t tag, char *desc)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv4 destination prefix and a mask length
`ifn`	Nexthop interface name; optional, but if specified nexthop address must be specified in `gate_str`
`gate_str`	Gateway: nexthop interface name or nexthop address
`distance`	Administrative distance
`tag`	Tag used as a “match” value to control redistribution via route maps
`desc`	Description of the static route

Output Parameters

None

Return Values

RIB_API_SET_ERR_INVALID_IPV4_ADDRESS when `p` does contain a valid IPv4 address

RIB_API_SET_ERR_INCONSISTENT_ADDRESS_MASK when the mask in `p` is not consistent with the address in `p`

RIB_API_SET_ERR_MALFORMED_GATEWAY when `gate_str` or `ifn` is not valid

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_NEXTHOP_OWN_ADDR when the next hop address is not valid

RIB_API_SET_ERR_MAX_STATIC_ROUTE_LIMIT when the maximum number of static routes has been reached

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_ipv6_route_prefix

The function creates a static route.

This function is called by the `ipv6 route` command.

Syntax

```
int
rib_cli_ipv6_route_prefix (struct rib_master *rm, vrf_id_t vrf_id,
                           struct prefix_ipv6 *p,
                           char *gate_str, int distance)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv6 destination prefix and a mask length
`gate_str`	Gateway: nexthop interface name or nexthop address
`distance`	Administrative distance

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_MALFORMED_GATEWAY when `gate_str` is not valid

RIB_API_SET_ERR_INVALID_IPV6_NEXTHOP_LINKLOCAL when an interface was not specified for a link-local nexthop

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_no_ip_route

The function deletes all IPv4 static routes for a given prefix in the default VRF (Virtual Routing and Forwarding) instance.

This function is called by the `no ip route` command.

Syntax

```
int
rib_cli_no_ip_route (struct rib_master *rm, vrf_id_t vrf_id,
                        struct prefix_ipv4 *p)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF instance identifier
`p`	Subnet: IPv4 destination prefix and a mask length

Output Parameters

None

Return Values

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_no_ip_route_all_vrf

The function deletes all IPv4 static routes for a given prefix in all VRF (Virtual Routing and Forwarding) instances.

This function is called by the `no ip route` command.

Syntax

```
int
rib_cli_no_ip_route_all_vrf (struct rib_master *rm, struct prefix_ipv4 *p)
```

Input Parameters

`rm`	RIB master
`p`	Subnet: IPv4 destination prefix and a mask length

Output Parameters

None

Return Values

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_no_ip_route_prefix

The function deletes a static route.

This function is called by the `no ip route` command.

You can specify:

- Only a nexthop interface name *or* nexthop address in `gate_str`
- Both a nexthop interface name in `ifn` *and* a nexthop address in `gate_str`

Syntax

```
int
rib_cli_no_ip_route_prefix (struct rib_master *rm,
                            vrf_id_t vrf_id, struct prefix_ipv4 *p,
                            char *ifn, char *gate_str, int distance,
                            u_int32_t tag, char *desc)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv4 destination prefix and a mask length
`ifn`	Interface name; optional, but if specified nexthop address must be specified in `gate_str`
`gate_str`	Gateway: nexthop interface name or nexthop address
`distance`	Administrative distance
`tag`	Tag used as a “match” value to control redistribution via route maps
`desc`	Description of the static route

Output Parameters

None

Return Values

RIB_API_SET_ERR_INVALID_IPV4_ADDRESS when `p` does contain a valid IPv4 address

RIB_API_SET_ERR_MALFORMED_GATEWAY when `gate_str` or `ifn` is not valid

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_cli_no_ipv6_route_prefix

The function deletes a static route.

This function is called by the `no ipv6 route` command.

Syntax

```
int
rib_cli_no_ipv6_route_prefix (struct rib_master *rm,
                              vrf_id_t vrf_id, struct prefix_ipv6 *p,
                              char *gate_str, int distance)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv6 destination prefix and a mask length
`gate_str`	Gateway: nexthop interface name or nexthop address
`distance`	Administrative distance

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_MALFORMED_GATEWAY when `gate_str` is not valid

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_fib_retain_set

The function sets the retention time for stale routes in the Forwarding Information Base (FIB) when `ribd` restarts.

This function is called by the `fib retain` command.

Syntax

```
int
rib_fib_retain_set (u_int32_t vr_id, int retain_time)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0
`retain_time`	Retention time in seconds or one of these constants from `ribd/ribd.h`:
`RIB_FIB_RETAIN_TIME_DEFAULT`	Retain stale routes for 60 seconds
`RIB_FIB_RETAIN_TIME_FOREVER`	Retain stale routes forever

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_ERR_INVALID_VALUE when `retain_time` is less than zero or greater than `RIB_FIB_RETAIN_TIME_MAX` (65535)

RIB_API_SET_SUCCESS when the function succeeds

rib_fib_retain_unset

The function sets the retention time for stale routes in the Forwarding Information Base (FIB) when `ribd` restarts to `RIB_FIB_RETAIN_TIME_DEFAULT` (60 seconds).

This function is called by the `no fib retain` command.

Syntax

```
int
rib_fib_retain_unset (u_int32_t vr_id)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_ip_vrf_isid_set

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

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

Syntax

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

Input Parameters

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

Output Parameters

None

Return Value

RIB_API_SET_ERR_VRF_NAME_TOO_LONG when `name` is more than 64 characters

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

RIB_API_SET_ERR_VRF_CANT_CREATE when a system resource limit is exceeded

RIB_API_SET_SUCCESS when the function succeeds

rib_ip_vrf_unset

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

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

Syntax

```
int
rib_ip_vrf_unset (struct ipi_vr *vr, char *name)
```

Input Parameters

`vr`	VRF instance
`name`	VRF name

Output Parameters

None

Return Value

RIB_API_SET_ERR_VRF_NOT_EXIST when `name` is not an VRF instance

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv4_route_set

The function creates a static route.

This function is called by the `ip route vrf` command.

Syntax

```
int
rib_ipv4_route_set (struct rib_master *rm,
                     vrf_id_t vrf_id, struct prefix_ipv4 *p,
                     union rib_nexthop *gate, char *ifname, int distance,
                     int metric, int snmp_route_type,
                     u_int32_t tag, char *desc)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv4 destination prefix and a mask length
`gate`	Gateway: nexthop address
`ifname`	Nexthop interface name
`distance`	Administrative distance
`metric`	Metric
`snmp_route_type`	SNMP route type; one of these constants from `lib/nexthop.h`:

`ROUTE_TYPE_LOCAL`

`ROUTE_TYPE_REMOTE`

`ROUTE_TYPE_OTHER`

`ROUTE_TYPE_REJECT`

`tag`	Tag used as a “match” value to control redistribution via route maps
`desc`	Description of the static route

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv4_route_stale_clear

The function clears stale IPv4 routes from the RIB (Routing Information Base) and FIB (Forwarding Information Base).

This function is called by the `clear ip route kernel` command.

Syntax

```
int
rib_ipv4_route_stale_clear (u_int32_t vr_id)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv4_route_unset

The function deletes a static route.

This function is called by the `no ip route vrf` command.

Syntax

```
int
rib_ipv4_route_unset (struct rib_master *rm,
                      vrf_id_t vrf_id, struct prefix_ipv4 *p,
                      union rib_nexthop *gate, char *ifname, int distance,
                      u_int32_t tag, char *desc)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv4 destination prefix and a mask length
`gate`	Gateway: nexthop address
`ifname`	Nexthop interface name
`distance`	Administrative distance
`tag`	Tag used as a “match” value to control redistribution via route maps
`desc`	Description of the static route

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv6_route_set

The function creates a static route.

Syntax

```
int
rib_ipv6_route_set (struct rib_master *rm,
                    vrf_id_t vrf_id, struct prefix_ipv6 *p,
                    union rib_nexthop *gate, char *ifname, int distance)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv6 destination prefix and a mask length
`gate`	Gateway: nexthop address
`ifname`	Nexthop interface name
`distance`	Administrative distance

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_INVALID_IPV6_NEXTHOP_LINKLOCAL when an interface was not specified for a link-local nexthop

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv6_route_stale_clear

The function clears stale IPv6 routes from the RIB (Routing Information Base) and FIB (Forwarding Information Base).

This function is called by the `clear ip route kernel` command.

Syntax

```
int
rib_ipv6_route_stale_clear (u_int32_t vr_id)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv6_route_unset

The function deletes a static route.

Syntax

```
int
rib_ipv6_route_unset (struct rib_master *rm,
                      vrf_id_t vrf_id, struct prefix_ipv6 *p,
                      union rib_nexthop *gate, char *ifname, int distance)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF (Virtual Routing and Forwarding) instance identifier
`p`	Subnet: IPv6 destination prefix and a mask length
`gate`	Gateway: nexthop address
`ifname`	Nexthop interface name
`distance`	Administrative distance

Output Parameters

None

Return Values

RIB_API_SET_ERR_VRF_NOT_EXIST when the VRF instance is not found

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv6_route_unset_all

The function deletes all static IPv4 route for a given VRF (Virtual Routing and Forwarding) instance and prefix.

Syntax

```
int
rib_ipv6_route_unset_all (struct rib_master *rm, vrf_id_t vrf_id,
                          struct prefix_ipv6 *p)
```

Input Parameters

`rm`	RIB master
`vrf_id`	VRF instance identifier

`p`	Subnet: IPv6 destination prefix and a mask length

Output Parameters

None

Return Values

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_ipv6_route_unset_all_vrf

The function deletes all static IPv6 routes for a given prefix from all VRF (Virtual Routing and Forwarding) instances.

Syntax

```
int
rib_ipv6_route_unset_all_vrf (struct rib_master *rm, struct prefix_ipv6 *p)
```

Input Parameters

`rm`	RIB master
`p`	Subnet: IPv6 destination prefix and a mask length

Output Parameters

None

Return Values

RIB_API_SET_ERR_NO_MATCHING_ROUTE when the static route does not exist

RIB_API_SET_SUCCESS when the function succeeds

rib_set_maximum_fib_routes

The function sets the maximum number of FIB (Forwarding Information Base) routes excluding kernel, connected, and static routes.

This function is called by the `max-fib-routes` command.

Syntax

```
int
rib_set_maximum_fib_routes (int vr_id, int num)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0
`num`	Maximum number of FIB routes, excluding kernel, connected, and static routes

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_set_maximum_static_routes

The function sets the maximum number of static routes.

This function is called by the `max-static-routes` command.

Syntax

```
int
rib_set_maximum_static_routes (int vr_id, int num)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0
`num`	Maximum number of static routes

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_ERR_MAX_STATIC_ROUTE_LIMIT when the number of configured static routes is already greater than `num`

RIB_API_SET_SUCCESS when the function succeeds

rib_unset_maximum_fib_routes

The function sets the maximum number of FIB (Forwarding Information Base) routes excluding kernel, connected, and static routes to `MAX_STATIC_ROUTE_DEFAULT` (4,294,967,294).

This function is called by the `no max-fib-routes` command.

Syntax

```
int
rib_unset_maximum_fib_routes (int vr_id)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

rib_unset_maximum_static_routes

The function sets the maximum number of static routes to `MAX_STATIC_ROUTE_DEFAULT` (4,294,967,294).

This function is called by the `no max-static-routes` command.

Syntax

```
int
rib_unset_maximum_static_routes (int vr_id)
```

Input Parameters

`vr_id`	Virtual Router identifier; for a non-Virtual-Router implementation, specify 0

Output Parameters

None

Return Values

RIB_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

RIB_API_SET_SUCCESS when the function succeeds

Index