ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Transparent Interconnection of Lots of Links
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 Transparent Interconnection of Lots of Links (TRILL).

Audience

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

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 14, *TRILL Management Information Base*
- Chapter 15, *Acronyms*

Related Documents

The following guides are related to this document:

- *Transparent Interconnection of Lots of Links Command Reference*
- *Transparent Interconnection of Lots of Links Configuration Guide*
- *Installation Guide*
- *Network Services Module Command Reference*
- *Network Services Module Developer Guide*
- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 Introduction

This chapter introduces ZebOS-XP TRILL (Transparent Interconnection of Lots of Links). TRILL supports the following:

- Single-instance of ISIS TRILL
- Up to eight distribution trees
- VLAN Pruning
- Multicast Pruning
- Backward compatible with IEEE 802.1 devices (such as hubs and bridges)

TRILL scalability is:

- 10–100 switches
- 0–1000 end stations in a given domain

TRILL Features

- MultiPath Forwarding—even distribution of traffic across available paths; maximum usage of available alternate pages
- Retention of state information—for faster convergence, stability and optimization of IP Multicast
- Minimize looping by decrementing the hop count
- Implementation of Routing Bridges

The TRILL module interfaces primarily with the NSM and IMI modules.

TRILL Protocol

The components of the TRILL protocol are as follows:

- Hello Protocol, the performance of which is based on:

  Number of Neighbors

  Hello-Interval

  Number of VLANs

  Hello Holding Interval
- DRB election, the performance of which is based on:

  Number of Neighbors

  Neighbor’s flapping
- AF appointments, the performance of which is based on:

  Number of Neighbors

  Number of End Station service VLANs
- SPF and DTree, the performances of which are based on:

  Number of RBridges in the domain

Number of DTrees

Number of adjacencies

References

- draft-ietf-trill-rbridge-af-04, “RBridges: Appointed Forwarders”
- draft-ietf-isis-TRILL-05, “TRILL Use of IS-IS”
- draft-ietf-TRILL-adj-07, “RBridges: Adjacency”
- draft-ietf-TRILL-rbridge-protocol-16, “RBridges: Base Protocol Specification”
- RFC 5556, “Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement
- draft-ietf-trill-rbridge-oam-02
- draft-ietf-trill-rbridge-mib-03.txt

CHAPTER 2 TRILL Software Architecture

This chapter describes the TRILL software architecture.

Overview

The TRILL module runs separately in ZebOS-XP. TRILL functionality is handled by the following modules:

- TRILL Core
- TRILL Interface Manager (IFM)
- TRILL L2 IS-IS Module
- TRILL Forwarding Module

TRILL Core

The TRILL core interfaces with external modules: IMI (Integrated Management Interface), NSM (Network Services Module) and HAL (Hardware Abstraction Layer).

The TRILL core is responsible for protocol initialization, interaction with user configuration, nickname selection, systemid generationlog messages, debug and interact with MIB objects. This module is responsible for Process Initialization, and interface with CLI commands and NSM, and interfaces primarily with the NSM and IMI modules. A summary follows:

- Initialize the TRILL process
- Handle notifications from NSM and CLI modules, including user requests such as configure, show and debug from CLI commands
- Handle the interface with NSM, and process events such as adding VLANs, Bridges, Interface up/down, etc.
- Handle the interface with HAL via NSM and program the forwarding layer

TRILL Interface Manager (IFM)

The TRILL Interface Manager (IFM) handles interface states and port states with other modules.

TRILL L2 IS-IS Module

The TRILL L2 IS-IS module encompasses the L2 IS-IS protocol for TRILL operations. A summary of using L2 IS-IS:

- Run TRILL Hello Protocol to establish adjacencies and elect Designated RBridge (DRB) on a broadcast link
- DRB then chooses Appointed Forwarders for VLANs, designated VLAN for inter RBridge communication, sends LSP on behalf of pseudonode, and issues CSNPs
- Run TRILL IS-IS protocol to exchange neighbor information and build a link state database; Generate SPF trees for unicast traffic and DTrees for multicast traffic
- Communicate TRILL-FDB tables to NSM, which then sends information to the hardware via HAL

This following sections describe the functions of the L2 IS-IS module, and its similarities and differences with the IS-IS module.

Collaboration

The L2 IS-IS module collaborates with the IMI module to acquire RBridge configuration, the NSM module to manage FDB, snoop on IGMP and MLD packets and to acquire interface/VLAN information. The L2 IS-IS module receives input from IMI/NSM modules:

IMI. Optional configurable parameters such as nickname, nickname Priority, nicknameDistributionTreeRootPriority, numberOfDistributionTrees, isAccessPort or isTrunkPort, isP2PHello, isMTUTestEnable

NSM. Interface states, IGMP and MLD messages, VLAN and interface information.

- The L2 IS-IS module computes the TRILL Route Tables that comprise of the following items and then sends that information to NSM:
- Unicast Route Table: The key is the RBridge nickname. The result consists of a hop count and a list of next-hops: egress interface; nickname of neighbor RBridge; MAC address of neighbor.
- DTree routing table: The key is Distribution tree root bridge nickname and VLAN. The result is a hop count and a list of paths (egress port, outer macDA, designated VLAN) on which the frame is to be sent. The information within the list is also used for adjacency checks. The ingress port should be one of the ports in that list.
- RPF check table: The key comprises of ingress RBridge nickname, VLAN and Distribution tree root bridge nickname. The result is the ingress port on which the frame is expected.

L2 IS-IS Frames

L2 IS-IS frames have a new Ethertype: “L2-IS-IS” (0x22F4).

The multicast destination address of All-IS-IS-RBridges is 01-80-C2-00-00-41.

IS-IS Instance

The TRILL IS-IS instance constitutes a single Level 1 IS-IS area using the fixed area-ID zero. This functionality consists of three components:

Hello Protocol and DRB Functions. The TRILL Hello protocol uses an IS-IS message known as “TRILL-hello”. This message starts with the same fixed header as an IS-IS LAN Hello, which includes the seven-bit priority for the router bridge (RBridge) to be the designated RBridge (DRB) on that link (broadcast). TRILL-Hellos are sent with the same timing as IS-IS LAN Hellos. Unlike IS-IS LAN Hellos, TRILL-hello elects only one DRB solely based on priority and MAC address. TRILL Hello frames are not padded.

TLVs. Each TRILL Hello frame contains a set for TLVs (Type, Length, Value); one TLV is mandatory.

- The mandatory special VLAN STLV carries the designated VLAN ID, a copy of the outer VLAN ID, Port ID, nickname, and flags indicating if the sender “assumes” it is the appointed forwarder (AF) for the VLAN and port on which the TRILL-hello was sent.
  - Multi-Topology Aware Port Capability TLV carries the same sub-TLV, as well as the optional sub-TLVs of enabled VLANs for end-station service; if DRB, appointed forwarder information.
  - If the sender is a DRB, TRILL Hellos are sent for each of the announcing VLANs and designated VLAN in the default case.
  - If the sender is not a DRB, TRILL hellos are sent tagged with each VLAN for which the RBridge is both the appointed forwarder and the designated VLAN.
  - In the mandatory sub-TLV, the following flags are determined:
    - if the sender is the appointed forwarder for this VLAN, the AF flag is set.
    - If the port is an access port, the AC flag is set.

    - If the port is trunk port, the TR flag is set.
    - If the VLAN mapping flag has been previously set in its data-structure, the VLAN-mapping flag is set in the hello frame.
    - If the sender is DRB and the corresponding flag in the data structure is set, the bypass pseudonode flag is set is the frame. The bypass pseudonode flag is also set when the number of adjacencies is less than two.
- Optional sub-TLVs are sent only via designated VLAN.
    - Designated VLANs are enabled for End-Station Service sub-TLV, which are sent by non-designated router bridges (non-DRB). This sub-TLV is sent if the enabled VLAN have been explicitly configured.
    - Appointed Forwarder (AF) sub-TLV are only sent by DRBs. Other router bridges (RB) on the link are informed they are the appointed forwarded for a VLAN set. If necessary, multiple Hellos are sent to cover all VLANs.
- Optional Neighbor TLVs list the MAC address and MTU of each neighbor.
    - If there is no adjacency, no TRILL Neighbor TLV is included in each Hello
    - If there are adjacencies, a Neighbor TLV is included in each Hello—a Neighbor TLV is only sent on a designated VLAN.
    - If the information of a Neighbor does not fit into one TLV, then multiple Hellos are delivered to cover all Neighbors.

Using TRILL Hello. Following is a summary of how TRILL Hello is used: .

- Elect DRB on a link.
- Choose appointed forwarders for the VLANs on a link.
- Detect VLAN-mapping which results in DRB appointing a single RBridge as VLAN forwarder for all VLANs.
- Each Hello frame can be a subset of the information it carries and will be processed as LSPs with CSNPs

Protocol Information Exchange Through LSPs. The L2 IS-IS module exchanges protocol related information with . other RBridges using LSP PDUs. This allows all RBridges to build their link-state database that represent the topology. From this database, the RBridge calculates SPF trees for unicast and multicast traffic to every other RBridge. L2 IS-IS uses the following TLVs in its linked state PDUs (LSP):

- Extended IS Reachability TLV is used to advertise the neighbors information containing their system ID and cost (metric).
- New sub-TLVs in Router Capability TLV are used to advertise information pertaining to nicknames and distribution trees:
    - The mandatory nickname sub-TLV contains the nickname, its priority, and the priority to become root of the distribution tree
    - Mandatory trees sub-TLV contain the information for computing trees:
        - The number of trees the RBridge is likely to use
        - The maximum number of trees the RBridge can calculate
        - The number of trees to compute
    - Optional: TRILL version sub-TLV
    - Optional: VLANs and spanning tree root sub-TLV containing a VLAN range for which the RBridge is appointed forwarder; the VLAN range specifies spanning tree roots, multicast router attached flag, and appointed forwarder status lost counter.

Tree Calculations. The SPF tree with Equal Cost Multi-Path (ECMP) for unicast is calculated from the link-state . database: which is same as for IS-IS. When building a forwarding table, an RBridge calculates shortest paths from itself. Nicknames are added into the shortest path calculation.

Distribution trees (DTrees) for multicast are calculated based on the tree information within Router Capability TLV exchanged in the LSPs. Each RBridge calculates trees listed in the Trees Identifiers sub-TLV of the RBridge with

highest root priority. In the absence of such sub-TLV, distribution trees with tree root being RBridges having first and highest tree root priorities are calculated.

TRILL Forwarding Module

The TRILL Forwarding module holds the configuration information for forwarding packets.

TRILL Configuration and Router Bridges

To initialize and compute paths, the TRILL module must be configured; default configuration is not provided. However, RBridges can start running with little configuration. The initial configuration requirements are as follows:

1. Create a TRILL bridge.
2. Create an RBridge instance and bind it with the TRILL bridge.
3. Enable the RBridge instance on the interface level.

Note that TRILL configuration is based on router bridges (RBridge). TRILL protocol running devices, network hubs and bridges, are referred to as RBridges. RBridges run a link state protocol and calculate shortest paths. IS-IS provides the transport to share the link-state information. Following is a summary of how frames are routed.

1. Native VLAN frames are encapsulated within an outer L2 header and a new TRILL header, and then transported to their destination.
2. The outer L2 header carries the address of the next-hop RBridge. The address is replaced at each hop until the frame reaches the destination RBridge.
3. The RBridge connected to the destination end-station unencapsulates the frame and forwards the native VLAN-frame to its destination end-station.
4. A hop-count in the TRILL header is decremented at each hop. The frame is discarded when the hop-count reaches zero, thus mitigating temporary loops.

CHAPTER 3 TRILL Data Structures

This chapter describes the data structures that support TRILL.

Common Data Structures

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

- `cli`
- `interface`

trill

Represents TRILL top data structure, and contains a Level-1 substructure. This structure contains the interface and global neighbor tables. It also contains system-wide configuration, including System ID, instance ID, Area Addresses, nickname database, self nickname list, distribution tree information and redistribute source information.

Definition

```
struct trill
{
  /* Tag of TRILL area.  */
  char *tag;

  /* Instance ID.  */
  u_int32_t instance_id;

  /* Pointer to TRILL master.  */
  struct trill_master *im;

  /* The bridge associated with this instance */
  struct trill_bridge* bridge;

  /* TRILL start time.  */
  pal_time_t start_time;

  /* TRILL System ID.  */
  u_char system_id[TRILL_SYSID_LENGTH];

 /* TRILL-type.  */
  /* this will default to L1 as TRILL runs only on one level */
  u_char is_type;

  /* TRILL protocols supported.  */
  /* - this proto type field will point to new prototype IS_L2 */
  u_char proto_type;
```

```
  /* TRILL administrative flags.  */
  u_char flags;
#define TRILL_FLAG_SHUTDOWN                     (1 << 1)

  /* Config flags.  */
  u_int16_t config;

#define TRILL_CONFIG_SYSTEM_ID                  (1 << 0)
#define TRILL_CONFIG_MINIMUM_MTU                (1 << 1)
#define TRILL_CONFIG_LSP_REFRESH_INTERVAL       (1 << 2)
#define TRILL_CONFIG_MAX_LSP_LIFETIME           (1 << 3)
#define TRILL_CONFIG_IGNORE_LSP_ERRORS          (1 << 4)
#define TRILL_CONFIG_FORWARD_DELAY              (1 << 5)
#define TRILL_CONFIG_ACCEPT_NON_ADJ             (1 << 6)
#define TRILL_CONFIG_UNICAST_ENABLE             (1 << 7)
#define TRILL_CONFIG_MULTICAST_ENABLE           (1 << 8)
#define TRILL_AUTOGEN_SYSTEM_ID                 (1 << 9)
#define TRILL_CONFIG_NUM_OF_MTU_PROBES          (1 << 10)


  /* Config variables.  */
  u_char max_area_addrs;                /* Number of Max Area Addresses. -  -
in trill this defaults to only one*/
  vector area_addrs;                    /* Manual Area Addresses. - - in trill
this should be only one addr*/
  vector recv_area_addrs;               /* Rcv Area Addresses - other than

     already in Manual area addr list. */
  u_int16_t lsp_refresh_interval;       /* LSP refresh interval. */
  u_int16_t max_lsp_lifetime;           /* MAX LSP Lifetime. */
  u_int32_t forward_delay;              /* Forward Delay */

  u_char confidence_local_recvd;        /* Confidence in local recvd frame */

  u_char confidence_remote_recvd;       /* Confidence in decapsulated frame */

  u_char confidence_static_config;      /* Confidence in static configuration
*/

  u_int16_t inter_rb_mtu;               /* originationLSP buffer size*/

  u_int16_t mtu_probe_counter;           /*mtu probe counter */

  u_char dtree_config;               /*Config flag for the dtree parameters */

  #define TRILL_CONFIG_MAX_RB_DTREES                (1 << 0)
  #define TRILL_CONFIG_NUM_DTREES_USE_CAMPUS        (1 << 1)
  #define TRILL_CONFIG_NUM_DTREES_TO_USE            (1 << 2)
  #define TRILL_CONFIG_NUM_DTREES_TO_COMPUTE        (1 << 3)

  u_int16_t num_of_dtrees_to_compute;   /* No of dtrees to compute */

  u_int16_t max_dtrees;                 /* Max number of d-trees the RB can
compute*/

  u_int16_t num_dtree_to_use;           /* No. of dtrees to use campus wide */
  /* Information per protocol.  */
```

```
  struct trill_proto proto[TRILL_PROTO_INDEX_MAX];

  /* Level context.  */
  struct trill_level level[TRILL_LEVEL_INDEX_MAX];

  /* Pointer to list of nicknames configured*/
  /*List of nicknames Auto generated and user configured {struct
trill_nickname}*/
  struct list *self_nickname;

  /*List of campus-wide nicknames  {struct trill_nickname}*/
  struct ls_table *nickdatabase;

  /*List of snooped multicast address  {struct trill_l2mcast}*/
  struct ls_table *l2mcasttable;

 /* List of appointed forwarder lost status per vlan for self RB*/
  struct ls_table *af_lost_table;

  /* Table to store info related to oam messages sent*/
  struct ls_table *chnl_echo_req_table;

  /* AF lost counter status for all LSPs, systemid + vlan(prefix) */
  struct ls_table *af_lost_table_lsp;

  /* List of pointers to data in af_lost_table_lsp, only whose afl counter is
   * changed.
   */
  struct list *afl_modified_list;

  /*List of nicknames sorted with higher priority tree root {struct
trill_nickname}*/
  struct list *treerootlist;

  /* List of configured dtrees to compute */
  struct list *dtrees_to_compute;

  /* List of configured dtrees to use */
  struct list *dtrees_to_use;

  /* Local circuit ID vector.  */
  vector circuit_vec;

 /* Tables.  */
  struct ls_table *if_table;            /* TRILL inteface table. */
  vector port_id_vec;                   /* TRILL interface vector */
                                        /* Index is used as port ID */
  struct ls_table *nbr_table;           /* Global neighbor table. */
  struct ls_table *nexthop_table;       /* Nexthop table. */

#if 0
  /* D-tree pruning related global tables */
  struct ls_table *l2_mcast_intf_table; /* L2 mcast group address table */
  struct ls_table *rbridge_vlan_bmp_mrt_table; /* Table of interested vlan
bitmaps */
  struct ls_table *l2_mcast_addr_table; /* table of l2 mcast group addresses
from GAADR tlv */
```

```
  struct trill_bridge_static_fdb_config *pstatic_fdblist;/* Pointer to Static-
FDB configuration */
  struct trill_bridge_static_dtree_config *pstatic_dtreelist;/* Pointer to
Static D-tree configuration */
  struct trill_snoop_table *table;/* Snoop table */
#endif
 /* TLVs.  */
  struct trill_tlv *tlv_area_addrs;      /* Area Addresses. */
  struct trill_tlv *tlv_protos;          /* Protocol supported. */
  /* -  to add more global level tlv;s here*/

  struct trill_tlv *tlv_group_addr;             /* Group Address TLV. */
  vector group_mac_addr_stlv;                   /* Group MAC Address Sub TLV*/

  vector router_capability_tlv;                 /* Router Capability TLV. */
  struct trill_tlv *stlv_nicknames;             /* Nicknames Sub TLV*/
  struct trill_tlv *stlv_tree_id;               /* Tree ID Sub TLV*/
  struct trill_tlv *stlv_tree_use_id;           /* Tree use ID Sub TLV*/
  struct trill_tlv *stlv_channel;               /* Channel Sub TLV*/
  /* Interested VLANs and Spanning Tree Root Sub TLV*/
  vector IntVLAN_stlv;

  /* Threads.  */
  struct thread *t_maxage_walker;       /* MaxAge walker. */
  struct thread *t_af_lost_status;       /* AF Lost Status. */

  /* Campus wide mtu */
  u_int16_t campus_wide_mtu;

  /* Minimum mtu */
  u_int16_t minimum_mtu;

  /* Number of mtu probes */

  u_int8_t mtu_probes;

 /*Forward delay*/
  u_int16_t fwd_delay;
  u_int16_t confidence_native;
  u_int16_t confidence_decap;
  u_int16_t confidence_static;
  u_int32_t max_nickname;

  #ifdef HAVE_SNMP
  /* Per vlan info table */
  struct ls_table *pvlan_table;

  /*Trill neighbor table based on MAC, for SNMP*/
  struct ls_table *trill_snmp_neigh;

  /*Trill Snooping Address table for SNMP */
  struct ls_table *trill_snmp_snoopaddr;

  /*Trill Snooping Port table for SNMP */
  struct ls_table *trill_snmp_snoopport;
  /*Trill vlan Port table for SNMP */
```

```
  struct ls_table *trill_snmp_vlanport;
  #endif /*HAVE_SNMP*/
  /*Timer values for OAM*/
  u_int16_t echo_req_interval;
  u_int16_t echo_req_timeout;

};
```

trill_level_proto

This data structure contains TRILL level-specific protocol information.

Definition

```
struct trill_level_proto
{
  /* Index of protocol. */
  u_char pindex;

  /* Flags. */
  u_char flags;
#define TRILL_SPF_DTREE_CALC                    (1 << 2)

  /* Back pointer to trill_level. */
  struct trill_level *il;

  /* Pointer to other level's level_proto. */

  /* SPF tree. */
  struct trill_vertex *spf;

  /* List of Multicast SPF Tree */
  vector dtree_vertex;        /*Tree number will be index & SPF_DTree will be
data*/
  /* Nickname list of dtrees above */
  vector dtree_nickname;
  /* Self vertex list in the dtrees above */
  vector self_vertex;

  u_int16_t orig_dtree;

  /* Suspended thread */
  struct thread *t_suspend;

 /* Current dtree root vertex */
  struct trill_vertex *rootv;

};
```

trill_interface

This data structure represents the data structure of the TRILL logical interface and contains a Level-1 substructure. This structure has a neighbor table for broadcast or point-to-point neighbor structure. It contains interface-related

configuration parameters. It also contains LSP flooding queue, sending-packet buffer, circuit-ID, read/write threads and other interface-related information.

Definition

```
struct trill_interface
{
  /* L2-IS-IS interface flags. */
  u_char flags;
#define TRILL_IF_UP                       (1 << 0)
#define TRILL_IF_DESTROY                  (1 << 1)

  /* Network type.  */
  u_char type;

  /* Local circuit ID assigned when interface is first created. */
  u_char circuit_id;

  /* port ID */
  u_int16_t port_id;

  /* IS-type & L2-IS-IS circuit-type. */
  u_char circuit_type;

  /* Interface lock. */
  int lock;

  /* Socket. */
  int sock;

  /* Socket for bpdu handling. */
  int bpdu_sock;

  /* BPDU bridge root ID */
  struct bridge_id *bpdu_bridge_id;

  /* Interface pointer passed by ZebOS-XP. */
  struct interface *ifp;

  /* back pointer to L2-IS-IS instance. */
  struct trill *top;

  /* Packet send buffer. */
  struct trill_fifo *obuf;

  /* Packet send buffer for channel. */
  struct trill_channel_fifo *channel_buf;

  union
  {
    /* LAN neighbor table. */
    struct ls_table *nbrs;

    /* P2P neighbor. */
    struct trill_neighbor *p2p_nbr;
  } u;
```

```
  /* LSP flood queue. */
  struct list *lsp_flood;

  /* Level for interface. */
  struct trill_if_level level[TRILL_LEVEL_INDEX_MAX];

  /* Configuration parameters. */
  struct trill_if_params *params;

/* Neighbor adjacency vector. */
  vector adjacency_vec;

  /* MTU Probe sequence number. */
  u_int16_t probe_seq_no;

  /* Threads. */
  struct thread *t_read;        /* PDU read thread. */
  struct thread *t_write;       /* PDU write thread. */
  struct thread *t_lsp_flood;   /* LSP flooding timer. */
  struct thread *t_lsp_rxmt;    /* P2P LSP Retransmit timer. */
  struct thread *t_bpdu_read;   /* BPDU read thread. */

  /* Uptime. */
  struct pal_timeval uptime;

  /* Statistics. */
  u_int32_t discarded;          /* discarded input count by error. */
  u_int32_t channeldiscarded;   /* counter for discarded channel frame */

};
```

trill_channel_packet

This data structure represents the data structure of the TRILL channel packets.

Definition

```
/* TRILL channel packet. */
struct trill_channel_packet
{
  /* Pointer of next packet. */
  struct trill_channel_packet *next;

  /* In/Out buffer. */
  struct stream *buf;

  /* Critcal extension TLV vector for Hop by Hop. */
  vector tlvvec_ce_hbh;

  /* Non-Critcal extension TLV vector for Hop by Hop. */
  vector tlvvec_nce_hbh;

  /* Critical extension TLV vector for IngresstoEgress. */
```

```
  vector tlvvec_ce_ie;

  /* Non-Critical extension TLV vector for IngresstoEgress. */
  vector tlvvec_nce_ie;

  /* Source MAC address. */
  u_char mac_src[ETHER_ADDR_LEN];

  /* Destination MAC address. */
  u_char mac_dst[ETHER_ADDR_LEN];

  /* Ingress nickname */
  u_int16_t ingress_nickname;

  /* Egress nickname */
  u_int16_t egress_nickname;

  /* Header options flags*/
  u_int32_t trill_op_header_flags;

  /* Header options extended flags*/
  u_int32_t trill_op_header_ext_flags;

 /* Option length. */
  u_int16_t oplength;
  u_int8_t hop_count;

    /* Outer vlan */
  u_int16_t ovlan;

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

};
```

trill_bridge

This data structure represents the data structure of the TRILL bridges.

Definition

```
struct trill_bridge{

  /* Housekeeping variables */
    struct trill_bridge *          next;
    struct trill_bridge * *        pprev;
    struct trill_port *            port_list;

    char       name[L2_BRIDGE_NAME_LEN+1];
    u_int32_t        instance_id;
    u_int8_t         type;
    /* VLAN tree indexed on vid. */
    struct route_table      *vlan_table;

    /* List of vlans added o the common instance */
    struct rlist_info *           vlan_list;

    /* Pointer to trill instance */
    struct trill  *trill_inst;

    /* Ageing timer,- this is maintained by the nsm module*/
    u_int16_t        ageing_timer;

    /* Forward timer -this is maintained by the nsm module */
    u_char        forward_delay;

    u_char                        is_default:1;

    s_int16_t                     num_ports;

};
```

trill_data_packet

This data structure represents the data structure of the TRILL data packets.

Definition

```
struct trill_data_packet
{
  /* In/Out buffer. */
  struct stream *buf;

  /* Critcal extension TLV vector for Hop by Hop. */
  vector tlvvec_ce_hbh;

  /* Non-Critcal extension TLV vector for Hop by Hop. */
```

```
    vector tlvvec_nce_hbh;

    /* Critical extension TLV vector for IngresstoEgress. */
    vector tlvvec_ce_ie;

    /* Non-Critical extension TLV vector for IngresstoEgress. */
    vector tlvvec_nce_ie;

    /* Source MAC address. */
    u_char mac_src[ETHER_ADDR_LEN];

    /* Destination MAC address. */
    u_char mac_dst[ETHER_ADDR_LEN];

    /* Ingress nickname */
    u_int16_t ingress_nickname;

    /* Egress nickname */
    u_int16_t egress_nickname;

    /* Header options flags*/
    u_int32_t trill_op_header_flags;

    /* Header options extended flags*/
    u_int32_t trill_op_header_ext_flags;

    /* Option length. */
    u_int16_t oplength;

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

  };
```

Databases

Definitions of the main TRILL databases are in the following sections.

Interface Table

Interface information is stored into the TRILL interface table that belongs to the TRILL instance (struct trill). Any TRILL enabled interface is stored in this table.

Neighbor Database

For broadcast interfaces, TRILL adjacency (TRILL neighbor) is kept in a table that belongs to the TRILL logical interface struct (struct trill_interface). For Point-to-Point interfaces, the neighbor structure is directly referenced because it is the only neighbor on that type of interface. In addition, the global adjacency table is kept in an TRILL instance structure (`struct trill`), to store neighbor information belonging to all interfaces.

LSP Database

The LSP database is the core of TRILL routing. All link-state information, known as Link State PDU, is advertised by neighbors in the same domain or area is stored in this database.

CHAPTER 4 TRILL NSM Interface API

This chapter describes the API that activates and deactivates the interfaces to the Network Services Module (NSM).

trill_activate_interface

This function activates the TRILL interface. It is contained in the file trill_api.c:

Syntax

```
s_int32_t
trill_activate_interface (struct trill_interface *trillif, char* name);
```

Input Parameters

`*trillif`	Specifies the instance of the TRILL interface.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_deactivate_interface

This function deactivates the TRILL interface. It is contained in the file trill_api.c:

Syntax

```
s_int32_t
trill_deactivate_interface (struct trill_interface *trillif, char* name)
```

Input Parameters

`*trillif`	Specifies the instance of the TRILL interface.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_nsm_if_state_up

This function brings up (enables) the specified interface. It is contained in the file trill_nsm.c.

Syntax

```
int
trill_nsm_if_state_up (struct interface *ifp);
```

Input Parameters

`*ifp`	Specifies the pointer to the interface structure.

Output Parameters

None

Return Values

0: Indicates the function call executed properly

trill_nsm_if_state_down

This function brings down (disables) the specified interface. It is contained in the file trill_nsm.c.

Syntax

```
s_int32_t
trill_nsm_if_state_down (struct interface *ifp);
```

Input Parameters

`*ifp`	Specifies the TRILL interface.

Output Parameters

None

Return Values

0: Indicates the function call executed properly.

CHAPTER 5 TRILL OAM

TRILL Operations Administration and Maintenance (OAM) is an interior gateway protocol which routes IP packets only within a single routing domain (autonomous system). OAM collects link-state data from routers to create a topology map of a network. This topology creates a routing table that is given to the internet layer. This routing table makes routing decisions based only on the destination IP address found in IP packets.

This chapter describes the API that is used for OAM-type configurations.

Overview

Whenever an LSP is received, both unicast OAM and D-tree OAM are scheduled along with pruning. A candidate list is a list data structure maintained and sorted based on distance. With OAM, this list is added to a binary heap data structure to help minimize the amount of time taken to insert and delete a vertex. AF calculation occurs all the time, even when there is no change in ES service VLAN. Whenever a hello message on a DVLAN is sent first, it also includes information used for AF calculation and appointments. Neighborship between the access port (TRILL traffic disable) and trunk port (native traffic disable) is not required, since this operation is done independently. Processing hellos on DVLANs helps avoid redundant and unnecessary processing.

Software Design

The following subsection describes the software design for TRILL SPF.

Multi-Destination OAM

Mission critical networks depend on the ability to proactively monitor and quickly troubleshoot defects and failures on a network. TRILL requires a well-defined OAM toolset to help protect the next generation of data forwarding technology in larger networks, such as data centers. ZebOS-XP supports two OAM tools:

- Multi-destination ping: checks connectivity between two or more RBridges along a particular distribution tree (D-Tree)
- Multi-destination traceroute: traces the multi-destination data path hop-by-hop to a target RBridge along a D-Tree

A channel-echo request message is created whenever ping or traceroute initiates from a CLI command. Ping and traceroute uses the same kind of frame for operation. To differentiate between ping and traceroute requests, and to properly match the response frames, a separate table is maintained at the Rbridge level. This table keeps track of every request message sent using the sequence number option provided in TRILL echo request/response and hopcount error frames. In addition, separate timers are dynamically created for every echo request sent. That is, whenever a new echo-request message is sent, a node is created and added to the table with new parameters.

Campus-Wide TRILL IS-IS MTU Size

In order to guarantee that packets are forwarded across all inter-RBridge links in a campus, an RBridge must know the size of link state information messages. In a stable campus, there must be agreement among all RBridges on the value of “Sz,” which is the minimum acceptable inter-RBridge link size for the proper operation of TRILL IS-IS. Sz is determined by having each RBridge advertise in its LSP its assumption of the value of the campus-wide Sz. This LSP element is known in IS-IS as the originatingLSPBufferSize TLV #14. By default, the campus MTU size is 1470 byte,

which limits the LSP buffer size. However, campus MTU can be configured and advertised in LSP, which allows RBridges to calculate the lowest campus MTU based on the information in LSP database.

Appointed Forwarders

TRILL supports multi-access LAN links that can have multiple end stations and RBridges attached. Where multiple RBridges are attached to a link, native traffic to and from end stations on that link is handled by a subset of those RBridges called “Appointed Forwarders” (AF). This allows native traffic on each VLAN to be handled by more than one RBridge.

When an RBridge receives a TRILL hello message asserting that the sender is the Appointed Forwarder, then the RBridge sets its VLAN inhibition timer for the link to the timer’s maximum value of the holding time in the received message. The RBridge must maintain VLAN inhibition timers for a link to which it connects to if it can offer end station service on that link, even if it is not currently Appointed Forwarder for any VLAN on that link. A “loop” may occur if two RBridges become AF for the same VLAN.

Trunk/P2P Port Mode Adjacency

When a multi-destination TRILL-encapsulated frame is received by an RBridge, a tree adjacency check is performed. This check may cause the frame to be discarded. Each RBridge RBn keeps a set of adjacencies ({ port, neighbor } pairs) for each D-Tree in its calculation. One of these adjacencies is towards the tree root RBi; the other is towards the leaves. RBridges drops a multi-destination frame that arrives at a port from an RBridge, which is not an adjacency for the tree on the frame being distributed.

When an RBridge RBn is no longer AF for a VLAN, the RB will forgets all end-station address information learned from decapsulating the VLAN native frames.

VLAN / Multicast Pruning

Pruning can prevent some unnecessary traffic from being circulated across the network. Pruning method is implemented on RBridges to prune VLANs from going to RBridges that do not have any hosts for that VLAN.

Receiving Channel Frames

The Layer-2 software forwarder “snoops” TRILL data frames to filter out TRILL channel frames. For TRILL channel frames, the frame is passed on to the TRILL module for further processing. Both the TRILL control frames and the TRILL channel frames are written to the same socket. In addition, at the receiving end of a socket, each frame has to be differentiated using TRILL data/control ether type and channel ether type.

When receiving a frame, a check is made to the ether type of the frame to differentiate between TRILL control frame and TRILL data frame. In addition, another check is made for channel ether type and it invokes the appropriate parser function.

Sending Channel Frames

When sending TRILL channel frames, the same socket must register with the TRILL control packet. Thus, a separate flow and queue for TRILL control and channel frame is maintained at each interface to send TRILL channel frames to the network. A thread is invoked to pop out trill-channel frames from this FIFO buffer and to write it on to the interface.

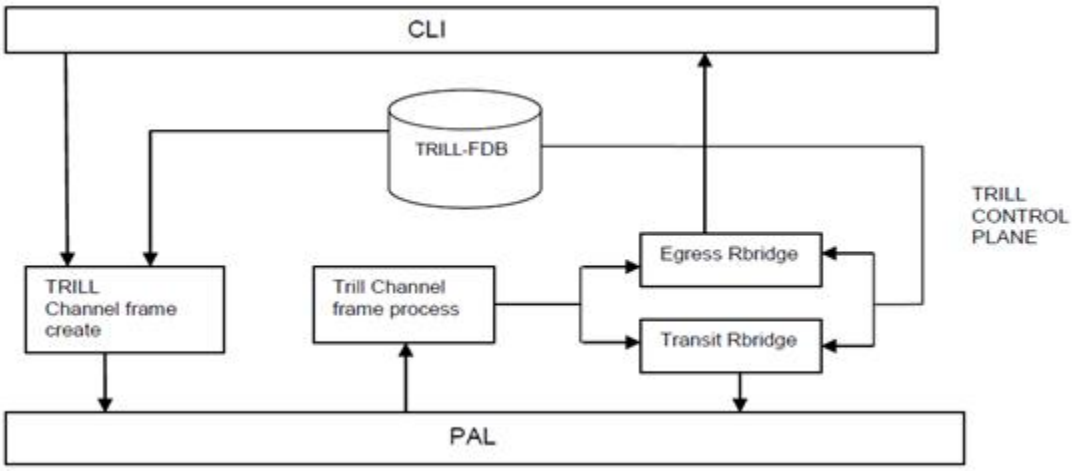

Figure 5-1: TRILL Channel Frame Process

Supported APIs

The following subsection describes the supported APIs for TRILL SPF.

trill_echo_request_send

This function creates an echo request packet and gets the nexthop info to transmit the frame to the egress rbridge. It is contained in the file trill_oam.c:

Syntax

```
int
trill_echo_request_receive (struct cli *cli,struct trill *top,
                            u_int16_t egress_nickname, u_int16_t hop_count,
                            u_int8_t flag, u_int16_t count)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`*top`	Specifies the pointer to the TRILL instance.
`egress_nickname`	Egress nickname to which ping echo request to be sent.
`hop_count`	Hop count to be set for the request frame.
`flag`	Flag to indicate whether request is send for ping or traceroute.
`count`	Number of echo request frame to be sent at a time.

Output Parameters

None

Return Values

TRILL_CHANNEL_RESULT_OK

TRILL_CHANNEL_FAIL

trill_parse_channel_frame

This function parses the channel packet. In addition, it checks whether a frame is intended for itself and, based on that, invokes the appropriate action to take.

Syntax

```
int
trill_parse_channel_frame (struct trill_channel_packet *dpkt,
                           struct trill_interface *isi)
```

Input Parameters

`*dpkt`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Values

0: Indicates the function call executed properly

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_frame_process_egress

This function processes channel frame intended for itself. In addition, it checks the protocol type and calls appropriate function to do further processing. It is contained in the file trill_oam.c:

Syntax

```
int
trill_channel_frame_process_egress (struct trill_channel_packet *dpkt,
                                    struct trill_interface *isi)
```

Input Parameters

`*dpkt`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Values

0: Indicates the function call executed properly.

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_frame_process_transit

This function processes channel frame that are not intended for itself In addition, it checks the hop count and forward the message properly. It is contained in the file trill_oam.c:

Syntax

```
int
trill_channel_frame_process_transit (struct trill_channel_packet *dpkt,
                                     struct trill_interface *isi)
```

Input Parameters

*dpkt	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Values

0: Indicates the function call executed properly.

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_process_hopcount_error

This function processes hop count errors. Based on the sequence number in the error frame, the function gets information regarding corresponding request frame send. Based on above info, this function creates a new echo request frame with incremented hop count if echo request was for traceroute functionality.

Syntax

```
int
trill_channel_process_hopcount_error (struct trill_channel_packet *pkt,
                                      struct trill_interface *isi, u_int32_t seq_num)
```

Input Parameters

*pkt	Specifies the pointer to the TRILL channel packet.
*isi	Pointer to the interface on which a frame was received.
seq_num	Sequence number of the received frame.

Output Parameters

None

Return Values

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_hopcount_error_send

This function creates and sends a hop count error frame. It is contained in the file trill_oam.c:

Syntax

```
int
trill_channel_hopcount_error_send (struct trill_channel_packet *dpkt,
                                   struct trill_interface *isi)
```

Input Parameters

*dpkt	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Values

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_unicast_frame_forward

This function forwards channel frames. Moreover, it find out the next hop info based on egress rbridge field of a received frame and forwards the frame accordingly after reducing hop count field. It is contained in the file trill_oam.c:

Syntax

```
int
trill_channel_unicast_frame_forward (struct trill_channel_packet *pkt,
                                     struct trill_interface *isi)
```

Input Parameters

*pkt	Specifies the pointer to the TRILL channel packet.
*isi	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Values

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_enable_oam_protocol_set

This function enables the OAM protocol.

Syntax

```
s_int32_t
```

```
trill_enable_oam_protocol_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_enable_oam_protocol_unset

This function disables the OAM protocol.

Syntax

```
s_int32_t
trill_enable_oam_protocol_unset(u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_enable_channel_protocol_set

This function enables the channel protocol.

Syntax

```
s_int32_t
trill_enable_channel_protocol_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_enable_channel_protocol_unset

This function enables the channel protocol.

Syntax

```
s_int32_t
trill_enable_channel_protocol_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_ping_interval_set

This function sets the ping interval.

Syntax

```
s_int32_t
trill_ping_interval_set (struct trill *top,u_int16_t val)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`val`	Echo request interval value.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_ping_interval_unset

This function disables the ping interval.

Syntax

```
s_int32_t
trill_ping_interval_unset (struct trill *top,u_int16_t val)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`val`	Echo request interval value.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_ping_timeout_set

This function sets the ping timeout value.

Syntax

```
s_int32_t
trill_ping_timeout_set (struct trill* top,u_int16_t timeout_val)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`val`	Echo request timeout value.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_ping_interval_unset

This function disables the ping timeout value.

Syntax

```
s_int32_t
```

```
trill_ping_timeout_unset (struct trill *top)
```

Input Parameters

*top	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_ping_interval_unset

This function disables the ping timeout value.

Syntax

```
s_int32_t
trill_ping_timeout_unset (struct trill *top)
```

Input Parameters

*top	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_channel_echo_reply_send

This function checks the parameters as part of a given flow chart and calls.

Syntax

```
int
trill_channel_echo_reply_send (struct trill_channel_packet *dpkt,
                               struct trill_interface *isi, u_int32_t seq_num)
```

Input Parameters

*dpkt	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.
seq_num	Specifies the pointer to the sequence number.

Output Parameters

None

Return Values

TRILL_CHANNEL_SUCCESS

TRILL_CHANNEL_FAIL

trill_lan_nsm_event

This function executes the NSM event process.

Syntax

```
int
trill_lan_nsm_event (struct thread *thread)
```

Input Parameters

`*thread`	Specifies the pointer to the thread value.

Output Parameters

None

Return Values

0

trill_p2p_nsm_event

This function TRILL point-to-point neighbor event handler.

Syntax

```
int
trill_p2p_nsm_event (struct thread *thread)
```

Input Parameters

`*thread`	Specifies the pointer to the thread value.

Output Parameters

None

Return Values

0

trill_ism_event

Depending on the specific event, this function changes the state accordingly and calls the corresponding event handler for the interface state value.

Syntax

```
int
trill_ism_event (struct thread *thread)
```

Input Parameters

`*thread` Specifies the pointer to the thread value.

Output Parameters

None

Return Values

0

trill_port_nsm_add

This function sends the TRILL port information to NSM.

Syntax

```
s_int32_t
trill_port_nsm_add (struct trill_if_level *ifl)
```

Input Parameters

`*ifl` Specifies the pointer to parent interface level.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_port_nsm_del

This function sends the TRILL port information to NSM.

Syntax

```
s_int32_t
trill_port_nsm_del (struct trill_if_level *ifl)
```

Input Parameters

`*ifl` Specifies the pointer to parent interface level.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_route_nsm_add

This function sends a unicast route add message to NSM.

Syntax

```
s_int32_t
trill_route_nsm_add (struct trill *top, struct trill_rtnhop *ir,
                     u_int32_t flags)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`*ir`	Specifies the TRILL route structure.
`flags`	Static or dynamic route.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_route_nsm_delete

This function sends a unicast route delete message to NSM.

Syntax

```
s_int32_t
trill_route_nsm_delete(struct trill *top, struct trill_rtnhop *ir,
                       u_int32_t flags)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`*ir`	Specifies the TRILL route structure.
`flags`	Static or dynamic route.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_port_nsm_del

This function sends the delete notification to NSM when port is removed from TRILL.

Syntax

```
s_int32_t
trill_port_nsm_del (struct trill_if_level *ifl)
```

Input Parameters

`*ifl`	Specifies the pointer to parent interface level.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_route_nsm_modify

This function sends modified route information

Syntax

```
s_int32_t
trill_route_nsm_modify (struct trill *top, struct trill_rtnhop *ir,
                        u_int32_t flags)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`*ir`	Specifies the TRILL route structure.
`flags`	Specifies the pointer to the interface flags.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_designated_vlan_nsm_add

This function sends the designated VLAN information to NSM.

Syntax

```
s_int32_t
trill_designated_vlan_nsm_add (struct trill_interface *isi, u_int16_t dvlan)
```

Input Parameters

`*ifl`	Specifies the pointer to parent interface level.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_af_vlan_nsm_add

This function sends the designated VLAN information to NSM.

Syntax

```
s_int32_t
trill_af_vlan_nsm_add (struct trill_if_level *ifl)
```

Input Parameters

`*ifl`	Specifies the pointer to parent interface level.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS

TRILL_API_SET_ERR_NO_NSM_CILENT

trill_mul_echo_request_send

This function creates a new echo request packet and gets the nexthop info to transmit the frame to the egress rbridge. In addition, it creates an echo_req_info node and updates the chnl_echo_req_table table. This function also starts the wait timer and adds an echo request frame to the output buffer.

Syntax

```
u_int32_t
trill_mul_echo_request_send (struct cli *cli, struct oam_mul_options *op,
                                    u_int16_t dtree, u_int8_t flag,
                                    u_int16_t instance_id)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`*op`	TRILL OAM init option.
`dtree`	Specifies the name of the DTree.
`flag`	Flag to indicate to send an echo request.
`instance_id`	Instance ID.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST

CLI_ERROR

CLI_SUCCESS

trill_parse_mul_ch_frame

This function parses channel packet and fills the trill_data_packet structure (refer to trill_data_packet on page 29). If the hopcount is greater than 1, it calls trill_channel_mul_frame_forward() to forward packet and return (refer to trill_channel_mul_frame_forward on page 53). If this is a ping request, it calls trill_process_mul_ping_request() (refer to trill_process_mul_ping_request on page 51). If this is a Traceroute request, it calls trill_process_mul_tracert_request() (refer to trill_process_mul_tracert_request on page 52).

Syntax

```
s_int32_t
trill_parse_mul_ch_frame (struct trill_channel_packet *dpkt,
                                 struct trill_interface *isi)
```

Input Parameters

`*dpk`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_mul_frame_forward

This function Forwards packet to all interfaces except the interface on which it was received.

Syntax

```
void
trill_channel_mul_frame_forward (struct trill_channel_packet *dpkt,
                                 struct trill_interface *isi)
```

Input Parameters

*dpk	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_process_mul_ping_request

This function will check parameters as per a given flow chart and calls trill_channel_echo_reply_send() (refer to trill_channel_echo_reply_send on page 53). This function may discard packets.

Syntax

```
u_int32_t
trill_process_mul_ping_request(struct trill_channel_packet * dpkt,
                               struct trill_interface *isi,
                               u_int32_t seq_num, u_int16_t instance_id)
```

Input Parameters

*dpk	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.
seq_num	Specifies the pointer to the sequence number.
instance_id	Instance ID.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_process_mul_tracert_request

This function checks parameters as per a given flow chart and calls trill_channel_echo_reply_send() or trill_channel_mul_hopcnt_err_send() (refer to trill_channel_echo_reply_send on page 53 and trill_channel_mul_hopcnt_err_send on page 52, respectively).

Syntax

```
u_int32_t
trill_process_mul_tracert_request (struct trill_channel_packet *dpkt,
                                   struct trill_interface *isi,
                                   u_int32_t seq_num, u_int16_t instance_id)
```

Input Parameters

`*dpk`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.
`seq_num`	Specifies the pointer to the sequence number.
`instance_id`	Instance ID.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_mul_hopcnt_err_send

This function forwards multicast hop count errors.

Syntax

```
u_int32_t
trill_channel_mul_hopcnt_err_send (struct trill_channel_packet *dpkt,
                                   struct trill_interface *isi,
                                   struct trill_nexthop *tmp_hopinfo)
```

Input Parameters

`*dpk`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.
`*hopinfo`	Specifies the hop count info.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_mul_frame_forward

This function forwards packet to all interfaces except the interface on which it was received.

Syntax

```
void
trill_channel_mul_frame_forward (struct trill_channel_packet *dpkt,
                                 struct trill_interface *isi)
```

Input Parameters

*dpk	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_channel_echo_reply_send

This function forwards echo replies.

Syntax

```
s_int32_t
trill_channel_echo_reply_send (struct trill_channel_packet *dpkt,
                               struct trill_interface *isi, u_int32_t seq_num,
                               u_int16_t egress_nickname, u_int16_t instance_id)
```

Input Parameters

*dpk	Received channel data packet.
*isi	Pointer to the interface on which a frame was received.
seq_num	Specifies the pointer to the sequence number.
egress_nickname	Egress nickname to which ping echo request to be sent.
instance_id	Instance ID.

Output Parameters

None

Return value

TRILL_CHANNEL_FAIL

TRILL_CHANNEL_SUCCESS

trill_vlan_inhibition_timer

This function is a VLAN inhibition timer handler. It is invoked when a VLAN inhibition timer expires. In addition, it updates interested VLAN sub-TLV and flood LSP and sends the updated AF information to NSM.

Syntax

```
int
trill_vlan_inhibition_timer (struct thread *thread)
```

Input Parameters

`*dpk`	Received channel data packet.
`*isi`	Pointer to the interface on which a frame was received.
`seq_num`	Specifies the pointer to the sequence number.
`egress_nickname`	Egress nickname to which ping echo request to be sent.
`instance_id`	Instance ID.

Output Parameters

None

Return value

0

trill_tlv_lspbuffsz_update

This function updates campus MTU TLV, creates the TLV and adds a TLV value. In addition, it updates LSP with the TLV.

Syntax

```
void
trill_tlv_lspbuffsz_update (struct trill *top)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return value

None

trill_rbridge_2way_restart_mtu_probe

This function restarts an MTU probing after moving the neighbor state machine to a 2-way state. In addition, it sends an MTU probe to all interfaces.

Syntax

```
void
```

```
trill_rbridge_2way_restart_mtu_probe (struct trill_if_level *ifl)
```

Input Parameters

`*ifl`	Specifies the pointer to parent interface level.

Output Parameters

None

Return value

None

trill_send_nickname_to_nsm

This function sends a nickname to NSM.

Syntax

```
s_int32_t
trill_send_nickname_to_nsm(struct trill *top, u_int16_t *name, u_int8_t flag)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`*name`	Specifies the name of the interface.
`flag`	Flag to indicate to send an echo request.

Output Parameters

None

Return value

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_API_SET_ERR_NICKNAME_DOESNOT_EXIST

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

TRILL_API_SET_SUCCESS

trill_forward_delay_nsm_notification

This function sends a delay notification to NSM.

Syntax

```
s_int32_t
trill_forward_delay_nsm_notification (struct trill *top, u_int16_t fwd_delay,
                                      u_int16_t nickname, u_int16_t vlan)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`fwd_delay`	Forward delay.

`nickname`	Specifies the nickname for the message.
`vlan`	Specifies the pointer to the VLAN.

Output Parameters

None

Return value

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_API_SET_SUCCESS

CHAPTER 6 TRILL Core Function API

The chapter contains the API that supports TRILL system and system components.

trill_systemid_set

This function sets the TRILL system ID for the specified instance. The system ID will be auto-generated by default. This function is called by the `systemid` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_systemid_set (u_int32_t vr_id, char *tag, char *systemid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.
`*systemid`	Specifies the system identification for the TRILL instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_SYSTEM_ID_CANT_CHANGED

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_systemid_unset

This function resets the TRILL system identification for the specified instance. The system ID is auto-generated. This function is called by the `no systemid` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_systemid_unset (u_int32_t vr_id, char *tag, char *systemid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the tag of the interface.
`*systemid`	Specifies the system identification for the TRILL instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_instance_set

This function creates the specified TRILL instance. It is called by the `rbridge trill` CLI command and is contained in the trill_api.c file.:

Syntax

```
s_int32_t
trill_instance_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INVALID_VALUE: Indicates invalid parameter was entered.

TRILL_API_SET_ERR_L2_INSTANCE_EXIST: Indicates the TRILL instance is already configured.

TRILL_API_SET_INSTANCE_EXCEED: Indicates the one allowed instance was already created.

trill_instance_unset

This function deletes the specified TRILL router instance. It is called by the `no rbridge trill` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_instance_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_instance_id_set

This function creates an instance ID. It is contained in the trill_api.c file.

Syntax

```
u_int32_t
trill_instance_id_set (struct trill *top)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

trill_instance_id_unset

This function unsets an instance ID. It is contained in the trill_api.c file.

Syntax

```
void
trill_instance_id_unset (struct trill *top)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

None

trill_instance_bridge_set

This function creates the specified TRILL RBridge instance and binds it with the bridge instance. It is called by the `rbridge trill bridge` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_instance_bridge_set (u_int32_t vr_id, char *name, char* bridgeid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the TRILL interface: 1–32.
`bridgeid`	Specifies the bridge instance: 1–32.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_BRIDGE_NOT_EXIST: Indicates the specified bridge does not exist; configure the bridge instance.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_instance_bridge_unset

This function resets (disables) the specified configuration. It is called by the `no rbridge trill bridge` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_instance_bridge_unset (u_int32_t vr_id, char *name, char *bridgeid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`bridgeid`	Specifies the bridge instance: 1–32.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_BRIDGE_NOT_EXIST: Indicates the specified bridge does not exist; configure the bridge instance.

TRILL_API_SET_ERR_BRIDGE_NOT_BINDED_TO_THIS: Indicates the specified bridge was not binded to an instance.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_bridge_add

This function adds a bridge to TRILL master:

- Create the VLAN tree.
- Verify the specified bridge name does not yet exist.
- Verify ports.

This function is contained in the trilld.c file.

Syntax

```
s_int32_t
trill_bridge_add (char * name, u_int8_t type, bool_t is_default)
```

Input Parameters

`name`	Specifies the name of the bridge.
`type`	Specifies the type of bridge.
`is_default`	Specifies the default bridge flag.

Output Parameters

None

Return Values

0: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

Possible errors:

- The length of the name of the specified bridge exceeds 60 characters.
- Unable to allocate memory for the bridge.
- The bridge needs to be added to the g_bridgelist.
- The bridge needs to be added to the TRILL master.

trill_bridge_delete

This function removes the bridge from TRILL.

- Change the status of each bridge port to inactive.
- Invalidate the forwarding database (FDB) generated from the bridge.
- Stop running LS IS-IS on the bridge level.

This function is contained in the trilld.c file.

Syntax

```
s_int32_t
trill_bridge_delete(char * name);
```

Input Parameters

`name`	Name description.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_bridge_vlan_add

This function adds a VLAN information to the specified bridge. It is contained in the trilld.c file.

Syntax

```
s_int32_t
trill_bridge_vlan_add (char *bridge_name, u_int16_t vid, char *vlan_name, u_int8_t
flags)
```

Input Parameters

*bridge_name	Specifies the name of the bridge.
vid	Specifies the VLAN ID to add: 1-4094.
*vlan_name	Specifies the VLAN name.
flags	Specifies the VLAN state, including:
suspended	(TRILL_VLAN_SUSPEND).
active	(TRILL_VLAN_ACTIVE).
configured	(TRILL_VLAN_INSTANCE_CONFIGURED).

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_bridge_vlan_delete

This function removes the VLAN from the bridge. It is contained in the trilld.c file.

Syntax

```
s_int32_t
trill_bridge_vlan_delete (char *bridge_name, u_int16_t vid);
```

Input Parameters

*bridge_name	Specifies the name of the bridge.
vid	Specifies the VLAN identification.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_nickname_set

This function sets set the nickname, its priority and the priority for a DTree root priority. It is called by the `nickname nickname-priority root-priority` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_nickname_set (struct cli *cli, u_int16_t nickname, u_int8_t priority,
                    u_int16_t root_priority)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`nickname`	Specifies the nickname for the DTree.
`priority`	Specifies the priority of the nickname: 128–255.
`root_priority`	Specifies the priority of the root: 0–65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_MAX_CONFIGURED_NICKNAME: Indicates the maximum number of nicknames (8) has been configured; no more nicknames can be configured.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available – Indicates memory failure: operation not performed.

TRILL_API_SET_ERR_SELF_NICKNAME_EXIST: Indicates the specified nickname was previously defined.

TRILL_API_SET_ERR_TAG_TOO_LONG: Indicates tag length exceeded 60 characters.

trill_nickname_unset

This function removes a specified nickname from the nickname list. It is called by the `no nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_nickname_unset (struct cli *cli, u_int16_t nickname)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`nickname`	Specifies the nickname to remove.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_MAX_CONFIGURED_NICKNAME: Indicates the maximum number of nicknames (8) has been configured; no more nicknames can be configured.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

TRILL_API_SET_ERR_SELF_NICKNAME_DOESNOT_EXIST: Indicates the specified nickname does not exist; there is no such nickname to remove.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

CHAPTER 7 TRILL IS-IS Command API

This chapter contains the Command API that supports TRILL IS-IS.

trill_if_link_type_set

This function sets the link type as broadcast or point-to-point. The default link type is broadcast. It is called by the `trill link-type` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_link_type_set(u_int32_t vr_id, char *name, s_int32_t type)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*name`	Specifies the interface name.
`type`	Specifies the link type: broadcast or point-to-point.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_ENABLED: Indicates the required interface was not enabled.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_INVALID_NETWORK_TYPE: Indicates invalid entry type was entered.

trill_if_pseudonode_set

This function enables the pseudonode flag. It is called by the `trill pseudonode enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_pseudonode_set (u_int32_t vr_id, char* name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_pseudonode_unset

This function disables the pseudonode flag. It is called by the `no trill pseudonode enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_pseudonode_unset (u_int32_t vr_id, char* name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_csnp_interval_set

This function sets the CSNP interval to the specified number of seconds. It is called by the `trill-isis csnp-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_csnp_interval_set (u_int32_t vr_id, char *name,
                            u_int32_t interval);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`internal`	Specifies the CSNP interval in seconds: 1-65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_csnp_interval_unset

This function resets the CSNP interval to the default value. It is called by the `no trill-isis csnp-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_csnp_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Default

10 seconds

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_hello_interval_set

This function sets the Hello interval to the specified seconds. It is called by the `trill-isis hello-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_hello_interval_set (u_int32_t vr_id, char *name,
                             u_int32_t interval)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`interval`	Specifies the hello interval in seconds: 1-65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_hello_interval_unset

This function resets the Hello interval to the default value. It is called by the `no trill-isis hello-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_hello_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Default

10 seconds

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_hello_interval_minimal_set

This function sets the flag for the Hello interval multiplier. It is called by the `trill-isis hello-interval minimal` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_hello_interval_minimal_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_hello_multiplier_set

This function sets the specified multiplier for Hello holding time. It is called by the `trill-isis hello-multiplier` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_hello_multiplier_set (u_int32_t vr_id, char *name,
                               u_int32_t multi);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`multi`	Specifies the multiplier value: 2–100.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_hello_multiplier_unset

This function resets the multiplier for Hello holding time to the default. It is called by the `no trill-isis hello-multiplier` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_hello_multiplier_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_metric_set

This function sets the specified interface metric. It is called by the `trill-isis metric` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_metric_set (u_int32_t vr_id, char *name, u_int32_t metric)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`metric`	Specifies the metric for the interface: 1–16777215.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_metric_unset

This function resets the specified interface metric to the default value. It is called by the `no trill-isis metric` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_metric_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_lsp_interval_set

This function sets the specified LSP transmission interval in milliseconds. It is called by the `trill-isis lsp-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_lsp_interval_set (u_int32_t vr_id, char *name, u_int32_t interval)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`interval`	Specifies the LSP transmission interval in milliseconds: 1–4294967295.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INVALID_VALUE: Indicates invalid parameter was entered.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_lsp_interval_unset

This function resets the LSP interval to default value: 33 seconds. It is called by the `no trill-isis lsp-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_lsp_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_priority_set

This function sets the specified port priority for the designation router selection. It is called by the `trill-isis port-priority` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_priority_set (u_int32_t vr_id, char *name, u_char priority)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`priority`	Specifies the port priority: 1-65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INVALID_VALUE: Indicates invalid parameter was entered.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_priority_unset

This function resets the port priority for designated router selection to the default value. It is called by the `no trill-isis port-priority` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_priority_unset(u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_inhibition_time_set

This function sets the inhibition interval for the database port. It is called by the `trill inhibition-time` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_inhibition_time_set(u_int32_t vr_id, char *name, u_int32_t interval)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.

`*name`	Specifies the instance ID.
`interval`	Specifies the inhibition time in seconds: 1-30.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

trill_inhibition_time_unset

This function resets the inhibition time for the database port to the default interval. It is called by the `no trill inhibition time` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_inhibition_time_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the instance ID.

Output Parameters

None

Default

30 seconds

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

trill_designated_vlan_set

This function sets the designated VLAN for the interface. It is called by the `trill designated-vlan` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_designated_vlan_set (u_int32_t vr_id, char *name, u_int16_t vlan)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`vlan`	Specifies the VLAN ID: 1-4094.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

RESULT_ERROR: Indicates an error occurred; operation not performed (the VLAN is not enabled).

trill_designated_vlan_unset

This function resets the VLAN to the default value. It is called by the `no trill designated-vlan` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_designated_vlan_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Default

1

Return Values

TRILL_API_SET_SUCCESS—Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_access_port_set

This function configures the specified a port as an access port. It is called by the `trill access-port enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_access_port_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_access_port_unset

This function resets the access port; the port is no longer an access port. It is called by the `no trill access-port` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_access_port_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_trunk_port_set

This function configures the specified port as a trunk port. It is called by the `trill trunk-port enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_trunk_port_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_trunk_port_unset

This function rests the specified trunk port; the port is no longer a trunk port. It is called by the `no trill trunk-port` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_trunk_port_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_end_station_service_set

This function sets the end station service VLAN. It is called by the `trill end-station-service-vlan` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_end_station_service_set (u_int32_t vr_id, char *name, u_int16_t vlan)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`vlan`	Specifies the VLAN range: 1–4094.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_end_station_service_unset

This function removes the end station service. It is called by the `no trill end-station-service-vlan` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_end_station_service_unset (u_int32_t vr_id, char *name, u_int16_t vlan)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`vlan`	Specifies the VLAN range: 1–4094.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

CHAPTER 8 TRILL IS-IS LSP Command API

This chapter contains the command API that supports the IS-IS LSP features.

trill_ignore_lsp_errors_set

This function sets TRILL to ignore LSP errors, which are pointed out by the checksum field. It is called by the `ignore-lsp-errors` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t trill_ignore_lsp_errors_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active Virtual Router instance

trill_ignore_lsp_errors_unset

This function configures the TRILL to not ignore all LSP errors, which are pointed out by the checksum field. It is called by the `no ignore-lsp-errors` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t trill_ignore_lsp_errors_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_lsp_gen_interval_set

This function sets the minimum interval between subsequent LSPs. Set the minimum interval between subsequent LSPs. This function is called by the `lsp-gen-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_lsp_gen_interval_set (u_int32_t vr_id, char *tag, u_char interval);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.
`interval`	Specifies the minimum time of the interval in seconds: 1–120.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_lsp_gen_interval_unset

This function resets the minimum interval between generating LSP to the default value. It is called by the `trill_lsp_gen_interval_unset` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_lsp_gen_interval_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Default

30 seconds.

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_lsp_refresh_interval_set

This function sets the LSP refresh interval between refreshes of locally generated LSPs. It is called by the `lsp-refresh-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_lsp_refresh_interval_set (u_int32_t vr_id, char *tag, u_int32_t interval)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.
`interval`	The timing of the refresh interval in seconds: 1–65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_lsp_refresh_interval_unset

This function resets the LSP refresh interval to the default value.

Syntax

```
s_int32_t
trill_lsp_refresh_interval_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Default

900 seconds.

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_max_lsp_lifetime_set

This function sets the maximum LSP lifetime for the added user. It is called by the `max-lsp-lifetime` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_max_lsp_lifetime_set (u_int32_t vr_id, char *tag, u_int32_t max_lifetime)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.
`max-lifetime`	Specifies the LSP lifetime in seconds: 350–65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_max_lsp_lifetime_unset

This function resets the LSP lifetime to the default value. It is called by the `no max-lsp-lifetime` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_max_lsp_lifetime_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Default

1200 seconds.

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_mcast_pruning_set

This function enables TRILL multicast pruning. It is called by the `mcast-pruning` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_mcast_pruning_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_mcast_pruning_del

This function disables TRILL multicast pruning.

Syntax

```
s_int32_t
trill_mcast_pruning_del (struct trill_rtnhop *ir, struct trill *top)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`*top`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

trill_mcast_pruning_unset

This function disables TRILL multicast pruning. It is called by the `no mcast-pruning` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_mcast_pruning_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_spf_interval_set

This function sets the interval for Shortest Path First (SPF) computations. It is called by the `spf-interval-exp` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_spf_interval_set (u_int32_t vr_id, char *tag,
                        u_int32_t min_delay, u_int32_t max_delay)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.
`mun_delay`	Specifies the minimum SPF interval in milliseconds: 0-2147483647.
`max_delay`	Specifies the maximum SPF interval in milliseconds: 0-2147483647.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INVALID_VALUE: Indicates invalid parameter was entered.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_spf_interval_unset

This function resets the Shortest Path First (SPF) interval to its default value.It is called by the `no spf-interval-exp` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_spf_interval_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_vlan_pruning_set

This function enables VLAN pruning. It is called by the `vlan-pruning` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_vlan_pruning_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_vlan_pruning_set

This function enables VLAN pruning. It is called by the `vlan-pruning` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_vlan_pruning_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_vlan_pruning_del

Syntax

```
s_int32_t
trill_vlan_pruning_del (struct trill_rtnhop *ir, struct trill *top)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

trill_vlan_pruning_unset

This function disables VLAN pruning. It is called by the `no vlan-pruning` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_vlan_pruning_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_update_vlan_pruning_table

Syntax

```
void
trill_update_vlan_pruning_table (struct trill_rtnhop *ir, u_int32_t ifindex,
                                 u_int16_t vlan)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`ifindex`	Specifies the name of the interface.
`vlan`	Specifies the name of the VLAN.

Output Parameters

None

Return Values

trill_update_mcast_pruning_table

Syntax

```
void
trill_update_mcast_pruning_table (struct trill_rtnhop *ir,
                                        u_int32_t ifindex,
                                        u_int16_t vlan, u_char *mac)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`ifindex`	Specifies the name of the interface.
`vlan`	Specifies the name of the VLAN.
`*mac`	Specifies the MAC address.

Output Parameters

None

Return Values

trill_if_retransmit_interval_set

This function sets the specified retransmission interval per LSP in seconds. It is called by the `trill-isis retransmit-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_retransmit_interval_set (u_int32_t vr_id, char *name,
                                  u_int32_t interval);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`interval`	Specifies the retransmission interval in seconds: 0-65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_IF_NOT_ENABLED: Indicates the required interface was not enabled.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_if_retransmit_interval_unset

This function resets the retransmission interval per LSP to the default value. It is called by the `no trill-isis retransmit-interval` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_if_retransmit_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.

Output Parameters

None

Default

5 seconds

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_IF_NOT_ENABLED: Indicates the required interface was not enabled.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_vlan_inhibition_timer

Syntax

```
int
trill_vlan_inhibition_timer (struct thread *thread)
```

Input Parameters

`*thread`	Specifies the pointer to the thread value.

Output Parameters

None

Return value

CHAPTER 9 TRILL RBridge Command API

This chapter contains the command API that supports RBridge-related features.

trill_rbridge_mtu_set

This function sets the campus MTU to be shared by the RBridge in its Hello packet. It is called by the `minimum-mtu` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_rbridge_mtu_set (u_int32_t vr_id, char *tag, u_int16_t rbridgemtu)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`rbridgemtu`	Specifies the RBridge MTU 1470-65535.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_num_mtu_probes_set

This function sets the number of MTU probes: the maximum attempts from RBridge before assessing the neighboring RBridge does not support the published MTU. The default number of MTU probes is 3. This function is called by the `number-of-mtu-probes` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_num_mtu_probes_set (u_int32_t vr_id, char* tag, u_int8_t mtuprobes)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`mtuprobes`	Specifies the maximum number of MTU probes to configure <1–255>. Default is 3.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_num_mtu_probes_unset

This function resets the number of mtu probes to the default value. It is called by the `no number-of-mtu-probes` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_num_mtu_probes_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_enable_mtu_probe_set

This function enables MTU probing. It is called by the `mtu-probe enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_enable_mtu_probe_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_enable_mtu_probe_unset

This function disables MTU probing. It is called by the `no mtu-probe enable` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_enable_mtu_probe_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_accept_nonadj_set

This function sets an RBridge to accept TRILL-encapsulated frames from the non-adjacent neighbor. The default value is false. This function is called by the `accept-non-adj` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_accept_nonadj_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_accept_nonadj_unset

This function disables accepting TRILL encapsulated data packets from the specified non-adjacent neighbor. It is called by the `no accept-non-adj` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_accept_nonadj_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_unicast_multicast_multipath_enable

This function enables multipathing a unicast or multicast packet. It is called by the `multipath` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_unicast_multicast_multipath_enable(u_int32_t vr_id, char *tag, bool_t isunicast)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`isunicast`	Specifies the flag to enable unicast/multicast.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_accept_NonAdj

Syntax

```
s_int32_t
trill_accept_NonAdj(struct trill *top, bool_t flag)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance
`flag`	Specifies the pointer to the interface flag.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

TRILL_API_SET_SUCCESS

trill_nsm_adjacency_update

Syntax

```
s_int32_t
trill_nsm_adjacency_update(struct trill_neighbor *nbr, u_int8_t flag)
```

Input Parameters

`*nbr`	Specifies the pointer to the TRILL instance
`flag`	Specifies the pointer to the interface flag.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_API_SET_SUCCESS

trill_unicast_multicast_multipath_disable

This function disables the multipath of the specified unicast or multicast packet. It is called by the `no multipath` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_unicast_multicast_multipath_disable(u_int32_t vr_id, char *tag, bool_t isunicast)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`isunicast`	Specifies the flag for enable unicast/multicast.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_set

This function sets a DTree for originating multicast traffic. It is called by the `originating-dtree` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_set (u_int32_t vr_id, char *tag, u_int16_t dtrees_to_compute)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtrees_to_compute`	Specifies the nickname for the DTree.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_unset

This function resets a DTree for originating multicast traffic. It is called by the `no originating-dtree` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_set

This function sets the number of dtrees to be computed by all rbridges. It is called by the `number-of-dtrees-to-compute` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_set (u_int32_t vr_id, char *tag, u_int16_t dtrees_to_compute)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtrees_to_compute`	Specifies the number of Dtrees to compute: 1-8.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_unset

This function resets all the values to default values. It is called by the `no number-of-dtrees-to-compute` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_tocompute_set

This function sets the DTrees (nickname) to be used by all RBridges in campus. It is called by the `dtree-tocompute` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_tocompute_set(u_int32_t vr_id, char *tag, u_int16_t dtree_nickname)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtree_nickname`	Specifies the DTree nickname to use.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates the specified entry already exists; remove the old entry.

TRILL_API_SET_ERR_NICKNAME_DOESNOT_EXIST: Indicates the specified nickname does not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_tocompute_unset

This function removes the specified nickname that was published to be computed by all campus RBridges. It is called by the `no dtree-nickname-to-compute` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
```

```
trill_dtree_tocompute_unset(u_int32_t vr_id, char *tag, u_int16_t dtree_nickname)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtree_nickname`	Specifies the DTree nickname to use.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_API_SET_ERR_NICKNAME_DOESNOT_EXIST: Indicates the specified nickname does not exist Indicates the specified nickname does not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_inuse_set

This function sets the DTrees to be used by this RBridge. It is called by the `dtree-in-use` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_inuse_set(u_int32_t vr_id, char *tag, u_int16_t dtree_nickname)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtree_nickname`	Specifies the DTree nickname to use: 1-0xFFFF.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_NICKNAME_DOESNOT_EXIST: Indicates the specified nickname does not exist.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_inuse_unset

This function removes the specified DTree that was set to be used by the specified RBridge. It is called by the `no dtree-in-use` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_inuse_unset(u_int32_t vr_id, char *tag, u_int16_t dtree_nickname)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtree_nickname`	Specifies the DTree nickname to use: 1–0xFFFF.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERROR: Indicates the specification was not configured.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_num_touse_set

This function sets the number of DTrees to be computed by the specified RBridge. It is called by the `number-of-dtrees-to-use` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_num_touse_set(u_int32_t vr_id, char *tag, s_int32_t dtree_num)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.
`dtree_num`	Specifies the DTree nickname to use: 1–8.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_num_touse_unset

This function resets the number of dtrees to be computed by this RBridge to the default value. It is called by the `no num-dtrees-touse` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_num_touse_unset(u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

rill_dtree_num_touse_unset

This function resets the number of dtrees to be computed by this RBridge to the default value. It is called by the `no num-dtrees-touse` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_dtree_num_touse_unset(u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Specifies the virtual router identification.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_dtree_nsm_add

Syntax

```
s_int32_t
trill_dtree_nsm_add (struct trill *top, struct trill_rtnhop *ir, u_int32_t flags)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance
`*ir`	Specifies the TRILL route structure.
`flags`	Specifies the pointer to the interface flags.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_dtree_nsm_del

This function ...

Syntax

```
s_int32_t
trill_dtree_nsm_del(struct trill *top, struct trill_rtnhop *ir, u_int32_t flags)
```

Input Parameters

`*top`	Specifies the identification of the virtual router.
`*ir`	Specifies the TRILL route structure.
`flags`	Specifies the pointer to the interface flags.

Output Parameters

None

Return Values

TRILL_API_SET_ERR_NO_NSM_CILENT

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_dtree_vlan_pruning_nsm_add

Syntax

```
s_int32_t
```

```
trill_dtree_vlan_pruning_nsm_add(struct trill_rtnhop *ir, struct trill *top)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_dtree_mcast_pruning_nsm_add

This function...

Syntax

```
s_int32_t
trill_dtree_mcast_pruning_nsm_add(struct trill_rtnhop *ir, struct trill *top)
```

Input Parameters

`*ir`	Specifies the TRILL route structure.
`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

TRILL_ERR_MEMORY_ALLOC_FAILURE

TRILL_API_SET_ERR_NSM_SEND_FAILED

trill_announcing_vlan_unset

This function resets the specified announcing VLAN. It is called by the `no trill announcing-vlan` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_announcing_vlan_unset(u_int32_t vr_id, char *name, u_int16_t vlan)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the interface.
`vlan`	Specifies the VLAN range: 1–4094.

Output Parameters

None

Return Values

TRILL_API_SET_ERROR: Indicates the specification was not configured.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

TRILL_API_SET_ERR_IF_PARAM_NOT_CONFIGURED: Indicates the configuration parameters for this interface do not exist.

trill_api_add_bridge_master

This function adds the bridge instances to the TRILL master. All TRILL specified bridges are added to the TRILL master.

Syntax

```
s_int32_t
trill_api_add_bridge_master (struct trill_bridge* bridge)
```

Input Parameters

`bridge`	Specifies the instance of the TRILL bridge structure.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_api_delete_bridge_master

This function removes the specified bridge instance from the TRILL master.

Syntax

```
s_int32_t
trill_api_delete_bridge_master (struct trill_bridge* bridge);
```

Input Parameters

`bridge`	Specifies the instance of the TRILL bridge structure.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_api_add_port

This function adds a port to a specified bridge instance.

Syntax

```
s_int32_t
trill_api_add_port (char* name, char *ifname);
```

Input Parameters

*name	Specifies the name of the bridge.
*ifname	Specifies the name of the interface.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_api_delete_port

This function deletes the port from the specified bridge instance.

Syntax

```
s_int32_t
trill_api_delete_port (char * name, char *ifname, u_int32_t ifindex)
```

Input Parameters

*name	Specifies the name of the bridge.
*ifname	Specifies the name of the interface.
ifindex	Specifies the interface index.

Output Parameters

None

Return Values

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_bridge_vlan_add_event

This function adds a VLAN event to a bridge and creates a new entry in the bridge.

Syntax

```
s_int32_t
trill_bridge_vlan_add_event (char* name, u_int16_t vid);
```

Input Parameters

`*name`	Specifies the name of the bridge.
`vid`	Specifies the VLAN identification.

Output Parameters

None

Return Values

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_bridge_vlan_delete_event

This function removes the specified VLAN event from the specified bridge.

Syntax

```
s_int32_t
trill_bridge_vlan_delete_event (char* name, u_int16_t vid);
```

Input Parameters

`*name`	Specifies the name of the bridge.
`vid`	Specifies the VLAN identification.

Output Parameters

None

Return Values

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_port_vlan_add_event

This function adds a VLAN to the specified port.

Syntax

```
s_int32_t
trill_port_vlan_add_event (u_int32_t vr_id, char *name, u_int32_t ifindex,
                           u_int16_t vid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the bridge.
`ifindex`	Specifies the interface index.
`vid`	Specifies the VLAN identification.

Output Parameters

None

Return Values

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_port_vlan_delete_event

This function removes the specified VLAN from a port.

Syntax

```
s_int32_t
trill_port_vlan_delete_event (u_int32_t vr_id, char *name, u_int32_t ifindex,
                              u_int32_t vid)
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*name`	Specifies the name of the bridge.
`ifindex`	Specifies the interface index.
`vid`	Specifies the VLAN identification.

Output Parameters

None

Return Values

RESULT_ERROR: Indicates an error occurred; operation not performed.

CHAPTER 10 TRILL Static FDB Command API

This chapter contains the command API that supports the static Forwarding Database (FDB).

trill_static_unicast_egress_set

This function adds a static route for unicast for reaching an egress router bridge (RBridge). It is called by the `add static unicast-trill egress-nickname` CLI command it is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_unicast_egress_set (struct cli *cli , u_int16_t nickname ,
                                 u_int16_t nxt_nickname , u_int16_t hopcount)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`nickname`	Specifies the nickname for egress.
`nxt_nickname`	Specifies the nexthop/neighbor nickname to reach the egress RBridge.
`hopcount`	Specifies the hopcount from the root: 1-255.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_API_SET_ERR_NBR_NICKINFO_NOEXIST: Indicates the mandatory nexthop nickname is missing from the Static Neighbor information table; the nickname needs to be added first.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: Indicates the tag length exceeds 60 characters.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_unicast_egress_unset

This function removes a static route for unicast for reaching an egress router bridge (RBridge). It is called by the `no add static unicast-trill egress-nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_unicast_egress_unset (struct cli *cli , u_int16_t nickname,
                                                     u_int16_t nxt_nickname)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`nickname`	Specifies the nickname for egress.
`nxt_nickname`	Specifies the nickname for the next egress.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_neighbor_macaddr_set

This function adds the specified static neighbor nickname: the port to reach it and the neighbor MAC address. It is called by the `add static fdb neighbor-nickname egress-interface mac-address` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_neighbor_macaddr_set (struct cli *cli, u_int16_t nickname,
                        u_char* interface ,u_int16_t ifindex, u_char* macaddr)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`nickname`	Specifies the nickname for the neighbor.
`interface`	Specifies the outgoing interface.
`ifindex`	Specifies the outgoing interface index.
`macaddr`	Specifies the neighbor MAC address.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NBR_NICK_EXIST: Indicates duplicate nexthop nickname; remove old entry from static neighbor table.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_neighbor_macaddr_unset

This function removes the specified static neighbor nickname, the port to reach it and the neighbor MAC address. It is called by the `no add static fdb neighbor-nickname X egress-interface` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_neighbor_macaddr_unset (struct cli *cli, u_int16_t nickname,
                                     u_char* interface)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`nickname`	Specifies the nickname of the neighbor.
`interface`	Specifies the name of the outgoing interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_ENTRY_IN_USE: Indicates specified entry in use by Static FBD.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_static_dtree_neighbor_adjacent_set

This function specifies each neighbor for the DTree. The neighbors are used for adjacency checks and setting downstream VLANs. It is called by the `add static multicast-fdb d-tree neighbor-nickname vlan-range` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_neighbor_adjacent_set (struct cli *cli, u_int16_t dtree ,
                                  u_int16_t neighbor_nck, u_int16_t start_vid,
                                  u_int16_t end_vid)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree`	Specifies the name of the DTree.
`neighbor_nck`	Specifies the nickname of the neighbor.
`start_vid`	Specifies the start of VLAN range.
`end_vid`	Specifies the end of VLAN Range

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

TRILL_API_SET_ERR_NBR_NICKINFO_NOEXIST: Indicates the mandatory nexthop Nickname is missing from the Static Neighbor information table; the nickname needs to be added first.

trill_static_dtree_neighbor_adjacent_unset

This function removes the adjacent neighbor port pair or the specified DTree. It is called by the `no add static multicast-fdb d-tree neighbor-nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_neighbor_adjacent_unset (struct cli *cli, u_int16_t dtree,
                                            u_int16_t neighbor_nck)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree`	Specifies the nickname of the DTree.
`neighbor_nck`	Specifies the nickname of the neighbor.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_dtree_neighbor_interface_rpf_set

This function is used for RPF checks. For each DTree and its incoming neighbor, specify the interface on which the frame is expected. It is called by the `add static multicast-fdb d-tree ingress-nickname ingress-interface` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_neighbor_interface_rpf_set (struct cli *cli,
                 u_int16_t dtree, u_int16_t ingress_nck, u_char* iface)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree`	Specifies the name of the DTree.

`ingress_nck`	Specifies the nickname of the ingress.
`iface`	Specifies the name of the incoming interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_static_dtree_neighbor_interface_rpf_unset

This function s used for RPF checks. Reset (remove) the reachable ingress nickname and interface for each DTree. It is called by the `no add static multicast-fdb d-tree ingress-nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_neighbor_interface_rpf_unset (struct cli *cli,
                                    u_int16_t dtree, u_int16_t ingress_nck)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree`	Specifies the name of the DTree.
`ingress_nck`	Specifies the nickname of the ingress.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_static_dstmac_vlan_set

This function sets the TRILL route for a native unicast MAC address and VLAN as follows:

- Sets the static MAC address of the destination.
- Sets the VLAN ID.
- Sets the egress RBridge to reach it.

This function is called by the add static l2-unicast-trill-fdb destination-mac vlan egress-nickname CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dstmac_vlan_set (struct cli *cli, u_char* mac_addr,
                              u_int16_t vlanid, u_int16_t egress_nck)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`mac_addr`	Specifies the destination MAC address.
`vlanid`	Specifies the VLAN associated with the destination MAC address (DMAC).
`egress_nck`	Specifies nickname of the egress.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_DST_MAC_EXIST: Indicates the specified MAC addresses was previously entered, remove the old MAC address from the Static MAC table.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_static_dstmac_vlan_unset

This function removes a specific TRILL route for a native unicast MAC address and VLAN, and removes the static MAC address. It is called by the `no add static l2-unicast-trill-fdb destination-mac vlan egress-nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dstmac_vlan_unset (struct cli *cli, u_char* mac_addr)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`mac_addr`	Specifies the destination MAC address.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

trill_static_dtree_multicast_set

This function statically adds the ingress DTree nickname to the FDB, which specifies if this tree is to be used for originating native multicast frames for a DTree. It is called by the `add static multicast-trill-fdb ingress-d-tree hop-count` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_multicast_set (struct cli *cli, u_int16_t dtree_nck,
                                  u_int16_t hop_count)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree_nck`	Specifies the name of the DTree.
`hop_count`	Specifies the hop count: 1–255.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_dtree_multicast_unset

This function resets (removes) the ingress DTree nickname to the FDB: the tree is no longer configured to originate native multicast frames for a DTree. It is called by the `no add static multicast-trill-fdb ingress-d-tree` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_dtree_multicast_unset (struct cli *cli, u_int16_t dtree_nck)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`dtree_nck`	Specifies the name of the DTree.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_NSM_SEND_FAILED: Indicates NSM was not sent.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_multicast_listener_set

This function adds a multicast listener for the specified neighbor. It is called by the `add static multicast-trill-fdb multicast-listener mcastmac-address nbr-nickname` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_multicast_listener_set (struct cli *cli,
                u_int16_t neighbor_nck, u_char* macaddr, u_int16_t vlan)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`neighbor_nck`	Specifies the nickname of the neighbor.
`macaddr`	Specifies the multicast MAC address.
`vlan`	Specifies a VLAN <1-4094>.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_ENTRY_EXIST: Indicates duplicated entry; remove old data entry.

TRILL_API_SET_ERR_TAG_TOO_LONG: The tag length exceeds 60 characters.

TRILL_ERR_MEMORY_ALLOC_FAILURE: Indicates memory is not available.

trill_static_multicast_listener_unset

This function statically resets (removes) the configured multicast MAC address with the associated downstream neighbor nickname. It is called by the `no add static multicast-trill-fdb multicast-listener mcastmac-address X:X:X nbr-nickname X` CLI command and is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_static_multicast_listener_unset (struct cli *cli,
                            u_int16_t neighbor_nck, u_char* macaddr)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`neighbor_nck`	Specifies the nickname of the neighbor.
`macaddr`	Specifies the multicast MAC address.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call worked properly.

TRILL_API_SET_ERR_NOEXIST: Indicates specified entry does not exist; no entry to remove.

TRILL_API_SET_ERR_NO_NSM_CILENT: Indicates the NSM client was not initialized.

TRILL_API_SET_ERR_TAG_TOO_LONG: Indicates The tag length exceeds 60 characters.

CHAPTER 11 TRILL ESADI

This chapter contains the API that supports End Station Address Distribution Information (ESADI).

System Overview

The ESADI protocol is part of the TRILL process. When ESADI is enabled on a global level per VLAN or set of VLANs, a TRILL LSP (Label Switch Path) will contain ESADI participation and ESADI unicast participation flag for all VLANs for which the RBridge is set as an Appointed Forwarder on any attached link (ESADI unicast participation flag bit is always set). The reserved bits in an interested VLAN for interested VLAN sub-TLV represent ESADI participation and unicast ESADI participation Bit. These bits are set if ESADI is enabled on an RBridge.

RBridge uses ESADI to announce and learn end station addresses rapidly and securely on VLANs. An RBridge that is announcing itself as connected to one or more VLAN (typically as an appointed forwarder device) and participates in the ESADI protocol is called an ESADI RBridge. The advantage of using ESADI is that the end station attachment information is not flooded to all RBridges through the core IS-IS instance, but only to participating RBridges advertising ESADI support for the VLAN in which those end stations occur. The advantages of ESADI include greater security and faster propagation of updates. It is also beneficial for policies that only use ESADI for end stations in some VLAN, so that it can then eliminate unknown unicast multi-destination frames.

System Processing

The following describes the ESADI protocol system processing as shown in Figure 11-1.

1. ESADI module is initialized using CLI (for example, `esadi enable`).
2. Trigger the update generation of TRILL-LSP. ESADI participation and ESADI unicast participation bits now enabled.
3. Schedule the self ESADI instance LSP.
4. Receive a neighbor ESADI Instance LSP.
5. Update an L2 table with end station MAC addresses and getting end station MAC addresses from the L2 table to be sent in ESADI Instance LSP.
6. Send and receive TRILL LSPs.
7. NSM notification to ESADI module of the L2 table being updated and the deleting of the learned L2 end station MAC entries when AF is lost.
8. CLI adds a static end station MAC address to simulate data plane L2 MAC table.
9. Fetch the AF info from TRILL to create an ESADI instance.
10. Unicast and pruned Dtree info is fetched by ESADI instances to send ESADI frames properly.

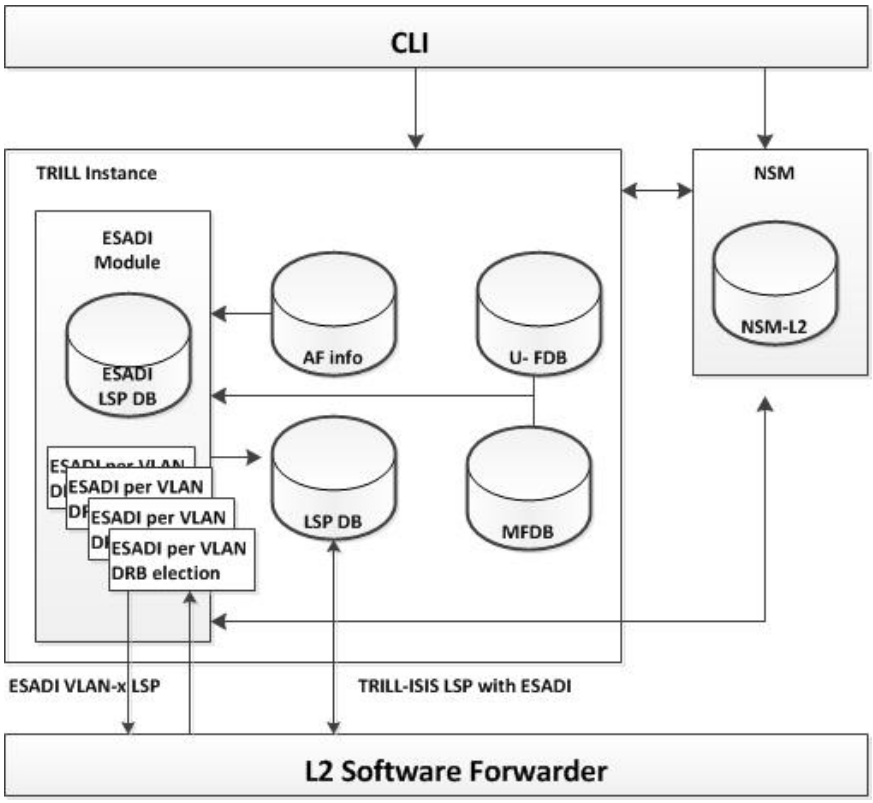

Figure 11-1: ESADI System Process

ESADI Initialization

The following describes the ESADI initialization process:

1. ESADI is enabled through the CLI.
2. An update is made on ESADI, including ESADI participation, the receiving of unicast ESADI PDU bits in interested VLANs and spanning tree roots sub-TLV for self LSPs, and the scheduling of LSPs for sending (if applicable).
3. For each AF VLAN in a TRILL instance (for all ports), create an ESADI instance and look for learned MAC addresses for that AF VLAN. For each ESADI instance, create an ESADI LSP (“frag 0”) and schedule the same to be sent on pruned dtree for that VLAN only if there exist a virtual neighbor for that VLAN (from TRILL LSPDB).
4. ESADI frame is sent on the pruned Dtree specified in the TRILL header with M bit set. The selection of a Dtree is done as following:
   - Get the list of Dtree names and interfaces from TRILL MFDB for VLAN on the ESADI frame that needs to be sent.
   - If Dtree to use is set for single computed Dtree, then choose that Dtree name.
   - If Dtree to use is set for multiple computed Dtrees, then first check if any of the self Dtree name is in the set. If yes, choose that Dtree name. Otherwise, choose the Dtree name with least path cost to root.
   - If no Dtree to use is mentioned in computed Dtrees, then choose the Dtree name with least path cost to root from the set.
5. For each ESADI instance, start the DRB election timer if at least one RB is interested for ESADI on that VLAN. In addition, update the count of expected neighbors per ESADI instance.

ESADI Neighborship and Adjacency Information

The following describes the ESADI Neighborship and Adjacency information process:

1. Start the DRB election when either the DRB election timer for an ESADI instance expires or all interested ESADI RBs LSPs are received (marked by the estimated neighbor count).
2. If no ESADI LSP for an ESADI instance is received, then move the instance to the down state. In the down state, if the ESADI instance exists, it will not send any ESADI PDUs for any change/update in its L2 entry table.
3. Schedule the ESADI CSNP if RB becomes DRB for an ESADI instance.

ESADI LSP DB Synchronization

The following describes the ESADI LSP database synchronization process:

1. When receiving an ESADI CSNP, an ESADI instance checks if it has all LSPs in its local database comparing with CSNP. If any of the LSPs is found to be missing, it will schedule ESADI PSNP for those LSPs.
2. When receiving an ESADI PSNP by a DRB, it will schedule the requested ESADI LSPs to be sent.
3. The ESADI instance DRB will send at least three ESADI CSNPs during each CSNP timer interval. If an ESADI instance has two or more ESADI neighbor and is not DRB and it receives no ESADI-CSNP PDUs for at least the CSNP time of the DRB, it may transmit an ESADI-CSNP.

Updation of ESADI LSP, Learned End Station Addresses and other Scenarios

The following describes the ESADI LSP, learned end station addresses and other scenarios process:

1. If a new ESADI neighbor is detected for a VLAN-x through receipt of ESADI-LSP-0, the recipient ESADI instance will immediately schedule its own ESADI-LSP-0 which may be unicast to the new RBridge.
2. If an ESADI RBridge loses an AF for a VLAN-x on its ports, ESADI instance does not participate in ESADI for that VLAN-x after sending a final ESADI-LSP, which “null” out its ESADI-LSP data (that is, removes all MACs of the connected end stations for that VLAN-x). It also schedules TRILL LSP with updated ESADI information in the interested VLANs and spanning tree roots sub-TLV for the self LSP.
3. If an ESADI LSP is received with zero MACs in the MAC Reachability TLV, the receiving ESADI instance will flush all the leaned L2 end station MAC addresses from that virtual neighbor on VLAN-x.
4. If an ESADI LSP is received with updated list of MAC Reachability TLV, the receiving ESADI instance will update the L2 table (delete / add) with end station addresses and also ESADI LSP in LSPDB.
5. Remove a neighbor's ESADI instance from the LSPDB when an RBridge becomes ISIS or data is unreachable, if any entry from the TRILL LSPDB is purged, or when a remote RBridge stops announcing ESADI participation.
6. TRILL triggers ESADI if a neighbor is removed from the ISIS-reachability TLV within a TRILL LSP. When removing a route, TRILL removes all learned end-station address from that neighbor, as well as the corresponding ESADI-LSPs from ESADI LSPDB of that RBridge and VLAN and the neighbor is DRB. It then reschedules the DRB election. If received RBridge is DRB for an ESADI instance, it will also schedule the updated CSNP.
7. If there is any update (add/delete) to end-station address table for a VLAN in self RBridge, there will be a trigger to that ESADI instance for updating and scheduling its ESADI LSP with updated end-station MACs.
8. If an access port link having AFs on RBridge running ESADI goes down (for example, unplugged), immediately send the self ESADI LSP with updated end station MACs for VLANs that were AFs on that port.

TRILL ESADI API

The following subsection lists and describes the API that supports TRILL ESADI.

trill_set_rbridge_esadi_status

This function sets the RBridge ESADI status.

Syntax

```
int
trill_set_rbridge_esadi_status (u_int32_t vr_id, u_int32_t instance,
                                u_int32_t vlanindex, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`val`	

Output Parameters

None

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_rbridge_esadi_status

This function gets the RBridge ESADI status.

Syntax

```
int
trill_get_rbridge_esadi_status (u_int32_t vr_id, u_int32_t instance,
                                u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_set_rbridge_esadi_confidence

This function sets the RBridge ESADI confidence.

Syntax

```
int
trill_set_rbridge_esadi_confidence (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t vlanindex, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`val`	

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID found

TRILL_API_SET_ERROR for invalid Instance ID not found

trill_get_rbridge_esadi_confidence

This function gets the RBridge ESADI confidence.

Syntax

```
int
trill_get_rbridge_esadi_confidence (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_set_rbridge_esadi_drbpriority

This function sets the designated RBridge ESADI priority.

Syntax

```
int
trill_set_rbridge_esadi_drbpriority (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t vlanindex, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`val`	

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID found

TRILL_API_SET_ERROR for invalid Instance ID not found

trill_get_rbridge_esadi_drbpriority

This function gets the designated RBridge ESADI priority.

Syntax

```
int
trill_get_rbridge_esadi_drbpriority (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_rbridge_esadi_drb

This function gets the designated RBridge ESADI information.

Syntax

```
int
trill_get_rbridge_esadi_drb (u_int32_t vr_id, u_int32_t instance,
                             u_int32_t vlanindex, u_char  **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`**ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_set_rbridge_esadi_drbholdingtime

This function sets the holding time for the designated RBridge.

Syntax

```
int
trill_set_rbridge_esadi_drbholdingtime (u_int32_t vr_id, u_int32_t instance,
                                        u_int32_t vlanindex, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`val`	

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID found

TRILL_API_SET_ERROR for invalid Instance ID not found

trill_get_rbridge_esadi_drbholdingtime

This function gets the holding time for the designated RBridge.

Syntax

```
int
trill_get_rbridge_esadi_drbholdingtime (u_int32_t vr_id, u_int32_t instance,
```

```
                                        u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_next_rbridge_esadi_status

This function gets the next status for the designated RBridge.

Syntax

```
int
trill_get_next_rbridge_esadi_status (u_int32_t vr_id, u_int32_t *instance,
                                     u_int32_t *vlanindex, int indexlen,
                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`indexlen`	Length of the index.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_next_rbridge_esadi_confidence

This function gets the next confidence for the designated RBridge.

Syntax

```
int
trill_get_next_rbridge_esadi_confidence (u_int32_t vr_id, u_int32_t *instance,
                                         u_int32_t *vlanindex, int indexlen,
                                         u_int32_t  *ret)
```


Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`indexlen`	Length of the index.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_next_rbridge_esadi_drbpriority

This function gets the next priority for the designated RBridge.

Syntax

```
int
trill_get_next_rbridge_esadi_drbpriority (u_int32_t vr_id, u_int32_t *instance,
                                          u_int32_t *vlanindex, int indexlen,
                                          u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`indexlen`	Length of the index.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`*ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_next_rbridge_esadi_drb

This function gets the next designated RBridge.

Syntax

```
int
trill_get_next_rbridge_esadi_drb (u_int32_t vr_id, u_int32_t *instance,
                                  u_int32_t *vlanindex, int indexlen,
                                  u_char  **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`indexlen`	Length of the index.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`**ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_get_next_rbridge_esadi_drbholdingtime

This function gets the next holding time for the designated RBridge.

Syntax

```
int
trill_get_next_rbridge_esadi_drbholdingtime (u_int32_t vr_id, u_int32_t *instance,
                                             u_int32_t *vlanindex, int indexlen,
                                             u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`indexlen`	Length of the index.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`**ret`	An integer containing a return value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID found

TRILL_API_GET_ERROR for invalid Instance ID not found

trill_esadi_module_init

This function initializes the ESADI protocol. It creates an ESADI instance and starts the DRB election timer for each ESADI instance.

Syntax

```
struct trill_esadi*
trill_esadi_module_init (struct trill *top)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Value

Null

trill_parse_esadi_frame

This function handles the ESADI PDUs and does basic validation (header).

Syntax

```
s_int32_t
trill_parse_esadi_frame (struct esadi_packet *epkt,
                         struct trill_interface *isi)
```

Input Parameters

`*epkt`	Received ESADI data packet
`*isi`	Pointer to the interface on which a frame was received.

Output Parameters

None

Return Value

RESULT_OK: Indicates the function call executed properly.

RESULT_ERROR: Indicates an error occurred; operation not performed.

trill_esadi_instance_lsp_process

This function processes the ESADI LSP (Label Switch Path) instance. If an ESADI instance exists for the RBridge and new neighbor is found on virtual link for that instance, then the instance is added it to the ESADI LSP database and an ESADI instance DRB election is then scheduled (if required). If the LSP contains no end station MAC addresses, then the function flushes all the learned layer 2 end station entries of that RBridge and VLAN and then updates NSM with layer 2 end station MAC addresses for that RBridge and VLAN.

Syntax

```
s_int32_t
trill_esadi_instance_lsp_process (struct esadi_packet *pkt,
                                  struct trill_if_level *ifl,
                                  struct esadi_instance *esadi_instance)
```

Input Parameters

`*pkt`	Received data packet
`*ifl`	Specifies the pointer to parent interface level.
`*esadi_instance`	

ESADI instance.

Output Parameters

None

Return Value

TRILL_PDU_LSP_ERROR

TRILL_PDU_LSP_ERR_BUFSIZE_MISMATCH

TRILL_PDU_LSP_ERR_INVALID_LIFETIME

TRILL_PDU_LSP_ERR_INSTANCE_EXIST

TRILL_PDU_LSP_ERR_CANT_ALLOCED

TRILL_PDU_SUCCESS

trill_esadi_instance_csnp_process

This function processes CSNP frames for an ESADI instance. If an ESADI LSP is missing from the local database, the function schedules an ESADI PSNP (Partial Sequence Numbers PDU) with any missing LSP. If a self LSP is missing in the received CSNP frame, then the function schedules a self ESADI LSP.

Syntax

```
s_int32_t
trill_esadi_instance_csnp_process (struct esadi_packet *pkt,
                                   struct trill_if_level *ifl,
                                   struct esadi_instance *esadi_instance)
```

Input Parameters

`*pkt` Received data packet

`*ifl` Specifies the pointer to parent interface level.

`*esadi_instance`

ESADI instance.

Output Parameters

None

Return Value

TRILL_PDU_LSP_ERROR

TRILL_PDU_CSNP_ERR_RECEIVED_NON_DRB

TRILL_PDU_CSNP_ERR_INVALID_PDU_LENGTH

TRILL_PDU_SUCCESS

trill_esadi_instance_psnp_process

This function processes PSNP frames for an ESADI instance. If there is a DRB for the ESADI instance, then the function only processes the PSNP frame and schedules the LSP reported in PSNP.

Syntax

```
s_int32_t
trill_esadi_instance_psnp_process (struct esadi_packet *pkt,
                                   struct trill_if_level *ifl,
                                   struct esadi_instance *esadi_instance)
```

Input Parameters

`*pkt`	Received data packet
`*ifl`	Specifies the pointer to parent interface level.
`*esadi_instance`	ESADI instance.

Output Parameters

None

Return Value

TRILL_PDU_LSP_ERROR

TRILL_PDU_CSNP_ERR_RECEIVED_NON_DRB

TRILL_PDU_CSNP_ERR_INVALID_PDU_LENGTH

TRILL_PDU_SUCCESS

trill_esadi_frame_send

This function sends ESADI PDUs to an interface. The Interface list will be received from the TRILL module for a VLAN and Dtree and will encapsulates the ESADI frame within the TRILL packet. In addition, it sets the m-bit in the TRILL header if the frame is sent on a multicast address. Finally, it sends the TRILL data frame on the TRILL interface socket.

Syntax

```
int
trill_esadi_frame_send (struct trill_interface *isi,
                        struct esadi_packet *pkt)
```

Input Parameters

`*isi`	Pointer to the interface on which a frame was received.
`*pkt`	Received data packet.

Output Parameters

None

Return Value

TRILL_PDU_LSP_ERROR

TRILL_PDU_CSNP_ERR_RECEIVED_NON_DRB

TRILL_PDU_CSNP_ERR_INVALID_PDU_LENGTH

TRILL_PDU_SUCCESS

trill_esadi_module_exit

This function gracefully disables the ESADI module on an RBridge. It updates the TRILL LSP with interested VLANs and spanning tree roots sub-TLV for self LSPs. In addition, it schedules any self LSP, flushes all entries in the ESADI LSP database, deletes the LSP database, deletes the ESADI instance list, and flushed all learned L2 end station MAC entries for AF (Appointed Forwarder) VLANs.

Syntax

```
void
trill_esadi_module_exit (struct trill *top)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Value

TRILL_PDU_LSP_ERROR

TRILL_PDU_CSNP_ERR_RECEIVED_NON_DRB

TRILL_PDU_CSNP_ERR_INVALID_PDU_LENGTH

TRILL_PDU_SUCCESS

trill_esadi_af_check

This function is called whenever there is a change in the aggregated AF bitmap (for all interfaces). It compare received aggregated bitmaps with the stored aggregated bitmaps. If AF is lost, it schedules the self ESADI instance LSP nulling the MACs and also bring down the ESADI instance. AF is new, create new ESADI Instance for new AF, schedule the ESADI LSP fragment 0, start DRB election timer and schedule the TRILL LSP.

Syntax

```
s_int8_t
trill_esadi_af_check (struct trill *top, u_int16_t vlan)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`vlan`	Specifies the pointer to the VLAN.

Output Parameters

None

Return Value

TRILL_ESADI_API_GET_ERROR

TRILL_ESADI_API_SET_SUCCESS

trill_esadi_participation_bits_notify

This function is called from whenever there is a change in the ESADI participation of the RBridge. It removes the neighbor entry from the ESADI LSPDB of a VLAN instance and flushes the L2 end station MAC entries. If DRBs bit is unset, this function then reschedules the DRB calculation for an ESADI instance.

Syntax

```
void
trill_esadi_participation_bits_notify (struct trill_lsp *lsp,
                                       u_int16_t vlan, u_int16_t ep_bits)
```

Input Parameters

`*top`	Specifies the pointer to the TRILL instance.
`vlan`	Specifies the pointer to the VLAN.
`ep_bits`	Status of the ESADI participation bit

Output Parameters

None

Return Value

TRILL_ESADI_API_GET_ERROR

TRILL_ESADI_API_SET_SUCCESS

CHAPTER 12 TRILL Show Command API

This chapter describes the API called by show commands.

trill_show_vlan_pruning

This function shows TRILL VLAN pruning Information. It is called by the `show trill pruning vlan` CLI command and is located in the trill_cli.c file.

Syntax

```
void
trill_show_vlan_pruning (struct cli *cli, struct trill *top,
                         u_int16_t dtree_name)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*top`	Specifies the pointer to the TRILL instance.
`dtree_name`	Specifies the dtree name.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_show_multicast_pruning

This function shows TRILL multicast pruning Information. It is called by the `show trill pruning multicast` CLI command and is located in the trill_cli.c file.

Syntax

```
void
trill_show_multicast_pruning (struct cli *cli, struct trill *top,
                              u_int16_t dtree_name)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*top`	Specifies the pointer to the TRILL instance.
`dtree_name`	Specifies the dtree name.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_cli_show_fdb

This function shows the details of the specified forwarding table: unicast, multicast or detail. It is called by the `show trill fdb` CLI command and is located in the trill_cli.c file.

Syntax

```
s_int32_t
trill_cli_show_fdb (struct cli *cli, char *level_str)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*level_str`	Specifies the forwarding table, including unicast, multicast or detail.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_show_lspdb

This function shows the TRILL link state database (LSPDB). Optionally, type, length, value (TLV) and sub-tlv can be shown. It is called by the `show trill detail` CLI command and is located in the trill_cli.c file.

Syntax

```
s_int32_t
trill_show_lspdb (struct cli *cli, char *level_str);
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*level_str`	Specifies the option to show TLV and sub-TLV.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_cli_show_neighbor

This function displays information from the TRILL neighbor table database. It is called by the `show trill neighbor` CLI command and is located in the trill_cli.c file.

Syntax

```
void
trill_cli_show_neighbor (struct cli* cli, struct trill* top);
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure.
`top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

CLI_SUCCESS

CLI_ERROR

trill_cli_show_interface

This function displays TRILL interface information: status of all interfaces of this RBridge. It is called by the `show trill interface` CLI command and is located in the trill_cli.c file.

Syntax

```
void
trill_cli_show_interface (struct cli *cli, struct interface *ifp);
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Values

None

trill_show_topology

This function displays the TRILL paths to all egress router bridges (RBridges) in the campus. It is called by the `show trill topology` CLI command and is located in the trill_cli.c file.

Syntax

```
s_int32_t
trill_show_topology (struct cli *cli, struct trill_level_proto *ilp);
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*ilp`	Specifies the pointer to the prototype structure.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_show_route

This function displays TRILL routes. It is located in the trill_cli.c file.

Syntax

```
void
trill_show_route (struct cli *cli, struct trill *top)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*ilp`	Specifies the pointer to the prototype structure.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_show_counter_level

This function displays the TRILL counter level. It is located in the trill_cli.c file.

Syntax

```
void
trill_show_counter_level (struct cli *cli, struct trill_level *il)
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`*il`	Specifies the pointer to the prototype structure.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

trill_cli_show_vlan_table

This function displays the TRILL VLAN tables. It is located in the trill_cli.c file.

Syntax

```
void
trill_cli_show_vlan_table(struct cli* cli , struct trill* top )
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure. Values includes 1-31.
`top`	Specifies the pointer to the TRILL instance.

Output Parameters

None

Return Value

CLI_SUCCESS

CLI_ERROR

CHAPTER 13 TRILL Clear and Debug Command API

This chapter provides information about the API called by clear and debug commands.

trill_proc_clear

This function stops the TRILL process and clears all TRILL databases. It is called by the `clear rbridge trill process` CLI command. This function is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_proc_clear (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*tag`	Specifies the identification of the RBridge instance.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_clear_counters

This function clears all TRILL counters. It is called by the `clear rbridge trill counter` CLI command. This function is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_clear_counters (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_INSTANCE_NOT_EXIST: Indicates there is no active IS-IS instance for the specified tag.

trill_clear_interface_counters

This function clears all TRILL interface counters. It is called by the `clear rbridge trill interface counter` CLI command. This function is contained in the trill_api.c file.

Syntax

```
s_int32_t
trill_clear_interface_counters (u_int32_t vr_id, char *ifname);
```

Input Parameters

`vr_id`	Specifies the identification of the virtual router.
`*ifname`	Specifies the name of the interface.

Output Parameters

None

Return Values

TRILL_API_SET_SUCCESS: Indicates the function call executed properly.

TRILL_API_SET_ERR_IF_NOT_EXIST: Indicates the specified interface is not present.

TRILL_API_SET_ERR_VR_NOT_EXIST: Indicates there is no active instance of a virtual router.

trill_debug_all_on

This function enables debugging for all TRILL modules or the specified TRILL module. It is called by the `debug trill` CLI command. This function is contained in the trill_cli.c file.

Syntax

```
void
trill_debug_all_on (struct cli *cli);
```

Input Parameters

`*cli`	Specifies the instance of the CLI structure, all modules or a specific module:

Output Parameters

None

Return Values

None

trill_debug_all_off

This function disables the debugging of TRILL modules. It is called by the no debug trill CLI command. This function is contained in the trill_cli.c file.

Syntax

```
void
trill_debug_all_off (struct cli *cli);
```

Input Parameters

*cli	Specifies the instance of the CLI structure, all modules or a specific module:

Output Parameters

None

Return Values

None

CHAPTER 14 TRILL Management Information Base

This chapter describes the TRILL Management Information Base (MIB) implemented in ZebOS-XP.

Overview

TRILL managed objects are accessed through a virtual information store using Simple Network Management Protocol (SNMP). It describes managed objects to configure and/or monitor TRILL for both single-hop and multi-hop sessions. ZebOS-XP modules act as the sub-agent that communicates with the master agent using AgentX protocol. Master Agent ideally runs in the agent. When SNMP Manager makes a request to the agent, the agent sends a corresponding request to the ZebOS-XP sub-agent through AgentX protocol. Similarly, the response sent by the sub-agent to the master agent is sent back to the manager by the agent.

For TRILL MIB objects, when an SNMP request arrives from the SNMP master agent to TRILL (which registers the TRILL MIB with the Master Agent), the TRILL module finds the corresponding data structures and accesses or updates the request. For a Get request, the object value is retrieved and sent to the master agent. For a Set request, the corresponding data structure is modified.

SNMP API

The following subsection lists the SNMP API for TRILL.

trill_get_rbridgebase_trill_version

This call gets rbridgeBaseTrillVersion; that is, the maximum version number of the TRILL protocol which this Rbridge instance supports.

Syntax

```
int
trill_get_rbridgebase_trill_version (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the version number.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_num_ports

This call gets rbridgeBaseNumPorts; that is, the number of ports controlled by this RBridge.

Syntax

```
int
trill_get_rbridgebase_num_ports (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the version number.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_forward_delay

This call gets modified aging time for address entries after an appointed forwarder change.

Syntax

```
int
trill_get_rbridgebase_forward_delay (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the version number.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridgebase_forward_delay

This call sets aging time for address.

Syntax

```
int
trill_set_rbridgebase_forward_delay (u_int32_t vr_id, u_int32_t instance,
```

```
                                        u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Aging time value to be set

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_unimultipath_enable

This call gets the status of unicast TRILL multipathing.

Syntax

```
int
trill_get_rbridgebase_unimultipath_enable (u_int32_t vr_id, u_int32_t instance,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the status of unicast TRILL multipathing.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridgebase_unimultipath_enable

This call sets status of unicast TRILL multipathing.

Syntax

```
int
trill_set_rbridgebase_unimultipath_enable (u_int32_t vr_id, u_int32_t instance,
                                           u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

`val`	Enables (1) or disables (2) status of unicast TRILL multipathing.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID found.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_multimultipath_enable

This call gets status of multidestination TRILL multipathing.

Syntax

```
int
trill_get_rbridgebase_multimultipath_enable (u_int32_t vr_id, u_int32_t instance,
                                             u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the status of unicast TRILL multipathing.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID .

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridgebase_multimultipath_enable

This call sets status of multicast TRILL multipathing.

Syntax

```
int
trill_set_rbridgebase_multimultipath_enable (u_int32_t vr_id,
                                             u_int32_t instance,
                                             u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Enables or disables status of multicast TRILL multipathing.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_nickname_number

This call gets number of nicknames this RBridge should have.

Syntax

```
int
trill_get_rbridgebase_nickname_number (u_int32_t vr_id, u_int32_t instance,
                                       u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the number of nicknames.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridgebase_nickname_number

This call sets number of nicknames this RBridge should have.

Syntax

```
int
trill_set_rbridgebase_nickname_number (u_int32_t vr_id, u_int32_t instance,
                                       int val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Number of nicknames that this Rbridge can have.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_accept_encapnonadj

This call gets whether to accept TRILL-encapsulated frames from a neighbor with which this RBridge does not have an IS-IS adjacency.

Syntax

```
int
trill_get_rbridgebase_accept_encapnonadj (u_int32_t vr_id, u_int32_t instance,
                                          u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the state of IS-IS adjacency.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridgebase_accept_encapnonadj

This call sets the state of IS-IS adjacency. (Accept TRILL-encapsulated frames from a neighbor with which this RBridge does not have an IS-IS adjacency.)

Syntax

```
int
trill_set_rbridgebase_accept_encapnonadj (u_int32_t vr_id, u_int32_t instance,
                                          u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	State of IS-IS Adjacency.
`0`	False
`1`	True

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridge_confidence_native

This call gets the confidence level associated with MAC addresses learned from native frames.

Syntax

```
int
trill_get_rbridge_confidence_native (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the confidence level associated with MAC addresses.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridge_confidence_native

This call sets the confidence level associated with MAC addresses learned from native frames.

Syntax

```
int
trill_set_rbridge_confidence_native (u_int32_t vr_id, u_int32_t instance,
                                     int val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Confidence level associated with MAC addresses.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridge_confidence_decap

This call gets the confidence level associated with inner MAC addresses learned after decapsulation of a TRILL data frame.

Syntax

```
int
trill_get_rbridge_confidence_decap (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the confidence level associated with MAC addresses.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridge_confidence_decap

This call sets the confidence level associated with inner MAC addresses learned after decapsulation of a TRILL data frame.

Syntax

```
int
trill_set_rbridge_confidence_decap (u_int32_t vr_id, u_int32_t instance,
                                    int val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Confidence level associated with inner MAC addresses.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridge_confidence_static

This call gets the confidence level associated with MAC addresses that are statically configured.

Syntax

```
int
trill_get_rbridge_confidence_static (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t  *ret)
```


Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the confidence level associated with MAC addresses that are statically configured.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridge_confidence_static

This call sets the confidence level associated with MAC addresses that are statically configured.

Syntax

```
int
trill_set_rbridge_confidence_static (u_int32_t vr_id, u_int32_t instance,
                                     int val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Confidence level associated with MAC addresses that are statically configured.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridge_dtree_priority

This call gets the distribution tree root priority for this Rbridge.

Syntax

```
int
trill_get_rbridge_dtree_priority (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the distribution tree root priority.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridge_dtree_priority

This call sets the distribution tree root priority for this Rbridge.

Syntax

```
int
trill_set_rbridge_dtree_priority (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Priority of rbridge dtree.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridge_dtree_activetrees

This call gets the total number of trees being computed by all Rbridges campus.

Syntax

```
int
trill_get_rbridge_dtree_activetrees (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the total number of trees being computed by all Rbridges campus.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridge_dtree_maxtrees

This call gets the maximum number of trees this Rbridge can compute.

Syntax

```
int
trill_get_rbridge_dtree_maxtrees (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the maximum number of trees this Rbridge can compute.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridge_dtree_desiredusetrees

This call gets the maximum number of trees this Rbridge would like to use for transmission of ingress multi-destination frames.

Syntax

```
int
trill_get_rbridge_dtree_desiredusetrees (u_int32_t vr_id, u_int32_t instance,
                                         u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the maximum number of trees this Rbridge would like to use.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridge_trillsz

This call gets the minimum acceptable inter-Rbridge link size for the campus for the proper operation of TRILL IS-IS.

Syntax

```
int
trill_get_rbridge_trillsz (u_int32_t vr_id, u_int32_t instance, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the minimum acceptable inter-Rbridge link size.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_get_rbridge_trill_minmtudesired

This call gets the desired minimum acceptable inter-RBridge link MTU for the campus, that is, originatingLSPBufferSize.

Syntax

```
int
trill_get_rbridge_trill_minmtudesired (u_int32_t vr_id, u_int32_t instance,
                                       u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the originatingLSPBufferSize.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_GET_ERROR for valid Instance ID not found.

trill_set_rbridge_trill_minmtudesired

This call sets the desired minimum acceptable inter-RBridge link MTU for the campus.

Syntax

```
int
trill_set_rbridge_trill_minmtudesired (u_int32_t vr_id, u_int32_t instance,
                                       u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.

`instance`	An integer that contains the TRILL instance ID.
`val`	OriginatingLSPBufferSize.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridge_trill_maxmtuprobes

This call gets the number of failed MTU-probes before the RBridge concludes that a particular MTU is not supported by a neighbor.

Syntax

```
int
trill_get_rbridge_trill_maxmtuprobes (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

Output Parameters

`ret`	Pointer to the number of failed MTU-probes.

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_set_rbridge_trill_maxmtuprobes

This call sets the number of failed MTU-probes.

Syntax

```
int
trill_set_rbridge_trill_maxmtuprobes (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`val`	Number of failed MTU-probes.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID.

TRILL_API_SET_ERROR for valid Instance ID not found.

trill_get_rbridgebase_nickname_priority

This call gets the RBridge's priority to hold this nickname.

Syntax

```
int
trill_get_rbridgebase_nickname_priority (u_int32_t vr_id, u_int32_t instance,
                                         u_int16_t nickname, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname for RBridge.

Output Parameters

`ret`	Pointer to the RBridge priority.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_get_rbridgebase_nickname_dtrpriority

This call gets the Distribution tree root priority for this nickname.

Syntax

```
int
trill_get_rbridgebase_nickname_dtrpriority (u_int32_t vr_id, u_int32_t instance,
                                            u_int16_t nickname, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname for RBridge.

Output Parameters

`ret`	Pointer to the Distribution tree root priority.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_get_rbridgebase_nickname_status

This call gets the status of the nick-name entry.

Syntax

```
int
trill_get_rbridgebase_nickname_status (u_int32_t vr_id, u_int32_t instance,
                                       u_int16_t nickname, u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
nickname	Nickname for RBridge.

Output Parameters

ret	Pointer to the status of the nick-name entry.
1	Static
2	Dynamic
3	Invalid

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_get_next_rbridgebase_nickname_priority

This call gets the RBridge's priority of the next nickname.

Syntax

```
int
trill_get_next_rbridgebase_nickname_priority (u_int32_t vr_id,
                                              u_int32_t *instance,
                                              u_int16_t *nickname,
                                              int index,
                                              u_int32_t *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
nickname	Nickname for RBridge.
indexlen	Length of the index

Output Parameters

`ret`	Pointer to the RBridge priority

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_get_next_rbridgebase_nickname_dtrpriority

This call gets the distribution tree root priority for next nickname.

Syntax

```
int
trill_get_next_rbridgebase_nickname_dtrpriority (u_int32_t vr_id,
                                                 u_int32_t *instance,
                                                 u_int16_t *nickname,
                                                 int index,
                                                 u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname for RBridge.
`indexlen`	Length of the index

Output Parameters

`ret`	Pointer to the Distribution tree root priority

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_get_next_rbridgebase_nickname_status

This call gets the status of next nick-name entry.

Syntax

```
int
trill_get_next_rbridgebase_nickname_status (u_int32_t vr_id,
                                            u_int32_t *instance,
                                            u_int16_t *nickname,
                                            int index,
                                            u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.

instance	An integer that contains the TRILL instance ID.
nickname	Nickname for RBridge.
indexlen	Length of the index

Output Parameters

ret	Pointer to the status of the nick-name entry.
1	Static
2	Dynamic
3	Invalid

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_GET_ERROR for valid Instance ID and Nickname not found.

trill_set_rbridgebase_nickname_priority

This call set the RBridge's priority to hold this nickname.

Syntax

```
int
trill_set_rbridgebase_nickname_priority (u_int32_t vr_id, u_int32_t instance,
                                         u_int16_t nickname, int  val)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
nickname	Nickname for RBridge.
val	RBridge's priority to be set.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_SET_ERROR for valid Instance ID and Nickname not found.

trill_set_rbridgebase_nickname_dtrpriority

This call sets the Distribution tree root priority for this nickname.

Syntax

```
int
trill_set_rbridgebase_nickname_dtrpriority (u_int32_t vr_id, u_int32_t instance,
                                            u_int16_t nickname, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname for RBridge.
`val`	Distribution tree root priority.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_SET_ERROR for valid Instance ID and Nickname not found.

trill_set_rbridgebase_nickname_status

This call set the status of the nick-name entry.

Syntax

```
int
trill_set_rbridgebase_nickname_status (u_int32_t vr_id, u_int32_t instance,
                                       u_int16_t nickname, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname for RBridge.
`val`	Nickname status
`1`	Static
`2`	Dynamic
`3`	Invalid

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Nickname.

TRILL_API_SET_ERROR for valid Instance ID and Nickname not found.

trill_get_rbridgebase_port_ifindex

This call gets the value of the instance of the ifIndex object.

Syntax

```
int
```

```
trill_get_rbridgebase_port_ifindex (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.

Output Parameters

ret	Pointer to the value of the instance of the ifIndex.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_disable

This call gets the status of port_disable.

Syntax

```
int
trill_get_rbridgebase_port_disable (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.

Output Parameters

ret	Pointer to the status of port_disable.
0	False
1	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_trunkport

This call gets the status of trunk_port.

Syntax

```
int
trill_get_rbridgebase_port_trunkport (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the status of port_trunk.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_accessport

This call gets the status of access_port.

Syntax

```
int
trill_get_rbridgebase_port_accessport (u_int32_t vr_id, u_int32_t instance,
                                       u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the status of port_trunk.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_p2phellos

This call gets the status of port, to know whether IS-IS P2P Hellos is supported or not.

Syntax

```
int
trill_get_rbridgebase_port_p2phellos (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t baseport, u_int32_t  *ret)
```


Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the status of port, to support IS-IS P2P hellos.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_state

This call gets the current state of port.

Syntax

```
int
trill_get_rbridgebase_port_state (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the state of port.
`1`	Uninhibited
`2`	portInhibited
`3`	vlanInhibited
`4`	Disabled
`5`	Broken

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_inhibitiontime

This call gets the time in seconds, that RBridge will inhibit forwarding on this port.

Syntax

```
int
trill_get_rbridgebase_port_inhibitiontime (u_int32_t vr_id, u_int32_t instance,
                                           u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the time in seconds

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_disablelearning

This call gets the status of disable learning of MAC addresses seen on this port.

Syntax

```
int
trill_get_rbridgebase_port_disablelearning (u_int32_t vr_id, u_int32_t instance,
                                            u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the status of disable learning.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_desireddesigvlan

This call gets the VLAN-ID that a DRB will specify in its TRILL-Hello.

Syntax

```
int
trill_get_rbridgebase_port_desireddesigvlan (u_int32_t vr_id, u_int32_t instance,
                                             u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the VLAN-ID.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_desigvlan

This call gets the VLAN-ID that the VLAN being used on this link for TRILL frames.

Syntax

```
int
trill_get_rbridgebase_port_desigvlan (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t baseport, u_int32_t *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the VLAN-ID.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_stproot

This call gets the bridge identifier of the root of the spanning tree.

Syntax

```
int
trill_get_rbridgebase_port_stproot (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t baseport, u_char  **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.

baseport	Port number for which this entry contains RBridge management information.

Output Parameters

ret	Pointer to the stproot bridge-id.
size	Length of the bridge-id

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_stprootchanges

This call gets the number of times a change in the root bridge is seen from spanning tree BPDUs received on this port.

Syntax

```
int
trill_get_rbridgebase_port_stprootchanges (u_int32_t vr_id, u_int32_t instance,
                                           u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.

Output Parameters

ret	Pointer to the root bridge counter.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridgebase_port_stpwiringcloset

This call gets the Bridge ID to be used as Spanning Tree root in BPDUs sent for the wiring closet topology solution.

Syntax

```
int
trill_get_rbridgebase_port_stpwiringcloset (u_int32_t vr_id, u_int32_t instance,
                                            u_int32_t baseport, u_char  **ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.

Output Parameters

`ret`	Pointer to the stproot bridge-id.
`size`	Length of the bridge-id

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_ifindex

This call gets the value of the instance of the next ifIndex object.

Syntax

```
int
trill_get_next_rbridgebase_port_ifindex (u_int32_t vr_id, u_int32_t *instance,
                                         u_int32_t *baseport, int indexlen,
                                         u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the value of the instance of the ifIndex.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_disable

This call gets the status of port_disable for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_disable (u_int32_t vr_id, u_int32_t *instance,
                                         u_int32_t *baseport, int indexlen,
                                         u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.

`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the status of port_disable.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_trunkport

This call gets the status of trunk_port for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_trunkport (u_int32_t vr_id, u_int32_t *instance,
                                           u_int32_t *baseport, int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the status of port_trunk.
`0`	False
`1`	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_accessport

This call gets the status of access_port for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_accessport (u_int32_t vr_id, u_int32_t *instance,
                                            u_int32_t *baseport, int indexlen,
                                            u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the status of port_trunk.
0	False
1	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_p2phellos

This call gets the status of next base port, to know whether IS-IS P2P Hellos is supported or not.

Syntax

```
int
trill_get_next_rbridgebase_port_p2phellos (u_int32_t vr_id, u_int32_t *instance,
                                           u_int32_t *baseport, int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the status of port, to support IS-IS P2P hellos
0	False
1	True

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_state

This call gets the current state of next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_state (u_int32_t vr_id, u_int32_t *instance,
                                       u_int32_t *baseport, int indexlen,
                                       u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the state of port.
1	Uninhibited
2	portInhibited
3	vlanInhibited
4	Disabled
5	Broken

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_inhibitiontime

This call gets the time in seconds, that RBridge will inhibit forwarding for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_inhibitiontime (u_int32_t vr_id,
                                                u_int32_t *instance,
                                                u_int32_t *baseport,
                                                int indexlen,
                                                u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the time in seconds.
1	Uninhibited

2	portInhibited
3	vlanInhibited
4	Disabled
5	Broken

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_disablelearning

This call gets the status of disable learning of MAC addresses seen for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_disablelearning (u_int32_t vr_id,
                                                 u_int32_t *instance,
                                                 u_int32_t *baseport,
                                                 int indexlen,
                                                 u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the status of disable learning.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_desireddesigvlan

This call gets the VLAN-ID that a DRB will specify in its TRILL-Hello.

Syntax

```
int
trill_get_next_rbridgebase_port_desireddesigvlan (u_int32_t vr_id,
                                                  u_int32_t *instance,
                                                  u_int32_t *baseport,
                                                  int indexlen,
                                                  u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the VLAN-ID.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_desigvlan

This call gets the VLAN-ID that the VLAN being used on this link for TRILL frames.

Syntax

```
int
trill_get_next_rbridgebase_port_desigvlan (u_int32_t vr_id, u_int32_t *instance,
                                           u_int32_t *baseport, int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the VLAN-ID.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_stproot

This call gets the bridge identifier of the root of the spanning tree for next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_stproot (u_int32_t vr_id, u_int32_t *instance,
                                         u_int32_t *baseport, int indexlen,
                                         u_char  **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the stproot bridge-id.
`size`	Length of the bridge-id

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_stprootchanges

This call gets the number of times a change in the root bridge is seen from spanning tree BPDUs received on next base port.

Syntax

```
int
trill_get_next_rbridgebase_port_stprootchanges (u_int32_t vr_id,
                                                u_int32_t *instance,
                                                u_int32_t *baseport,
                                                int indexlen,
                                                u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the root bridge counter.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_get_next_rbridgebase_port_stpwiringcloset

This call gets the Bridge ID to be used as Spanning Tree root in BPDUs sent for the Wiring Closet topology solution.

Syntax

```
int
trill_get_next_rbridgebase_port_stpwiringcloset (u_int32_t vr_id,
                                                 u_int32_t *instance,
                                                 u_int32_t *baseport,
                                                 int indexlen,
                                                 u_char  **ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the stproot bridge-id.
size	Length of the bridge-id

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_GET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_disable

This call sets the status of port_disable.

Syntax

```
int
trill_set_rbridgebase_port_disable (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t baseport,u_int32_t val)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
val	Status of port_disable.
0	False
1	True

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_trunkport

This call sets the status of port to support trunk.

Syntax

```
int
trill_set_rbridgebase_port_trunkport (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t baseport, u_int32_t val)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
val	Status of port_trunk.
0	False
1	True

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_accessport

This call sets the status of port to support access port feature.

Syntax

```
int
trill_set_rbridgebase_port_accessport (u_int32_t vr_id, u_int32_t instance,
                                       u_int32_t baseport, u_int32_t val)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
baseport	Port number for which this entry contains RBridge management information.
val	Status of port_trunk.
0	False
1	True

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_p2phellos

This call sets the status of port, to know whether IS-IS P2P Hellos is supported or not.

Syntax

```
int
trill_set_rbridgebase_port_p2phellos (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t baseport, u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`val`	Status of port, to support IS-IS P2P hellos.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_inhibitiontime

This call sets the time in seconds, that RBridge will inhibit forwarding on this port.

Syntax

```
int
trill_set_rbridgebase_port_inhibitiontime (u_int32_t vr_id, u_int32_t instance,
                                           u_int32_t baseport, u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`val`	Time in seconds.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_disablelearning

This call sets the status of port to support disable learning of MAC addresses.

Syntax

```
int
trill_set_rbridgebase_port_disablelearning (u_int32_t vr_id, u_int32_t instance,
                                            u_int32_t baseport, u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`val`	Status of port to support disable learning

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_desireddesigvlan

This call sets the VLAN-ID that a DRB will specify in its TRILL-Hello.

Syntax

```
int
trill_set_rbridgebase_port_desireddesigvlan (u_int32_t vr_id,
                                             u_int32_t instance,
                                             u_int32_t baseport,
                                             u_int32_t val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`val`	VLAN-ID

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_set_rbridgebase_port_stpwiringcloset

This call sets the Bridge ID to be used as Spanning Tree root in BPDUs sent for the Wiring Closet topology solution.

Syntax

```
int
trill_set_rbridgebase_port_stpwiringcloset (u_int32_t vr_id, u_int32_t instance,
                                            u_int32_t baseport, int length,
                                            u_char *val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	Port number for which this entry contains RBridge management information.
`val`	Pointer to the bridge-id used as spanning tree root in BPDU.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID and Port Number.

TRILL_API_SET_ERROR for valid Instance ID and Port Number not found.

trill_get_rbridge_unifdb_nick

This call gets the RBridge nickname which is placed in the Egress Nickname field of a TRILL frame.

Syntax

```
int
trill_get_rbridge_unifdb_nick (u_int32_t vr_id, u_int32_t instance,
                               u_int32_t fdbid, u_char *macaddr,
                               u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`fdbid`	Identity of this filtering database.
`macaddr`	Unicast MAC address for which the device has forwarding information.

Output Parameters

`ret`	RBridge nickname

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_rbridge_unifdb_confidence

This call gets the confidence level associated with this entry.

Syntax

```
int
trill_get_rbridge_unifdb_confidence (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t fdbid, u_char *macaddr,
                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`fdbid`	Identity of this filtering database.
`macaddr`	Unicast MAC address for which the device has forwarding information.

Output Parameters

`ret`	Confidence level.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_rbridge_unifdb_status

This call gets status of this entry.

Syntax

```
int
trill_get_rbridge_unifdb_status (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t fdbid, u_char *macaddr,
                                 u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`fdbid`	Identity of this filtering database.
`macaddr`	Unicast MAC address for which the device has forwarding information.

Output Parameters

`ret`	Status of UniFdb.

1	Other
2	Invalid
3	Learned
4	Self
5	Mgmt
6	ESADI

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_rbridge_unifdb_port

This call gets the port number of the port on which a frame having a source address equal to the value of the corresponding instance of rbridgeUniFdbAddress has been seen.

Syntax

```
int
trill_get_rbridge_unifdb_port (u_int32_t vr_id, u_int32_t instance,
                               u_int32_t fdbid, u_char *macaddr,
                               u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
fdbid	Identity of this filtering database.
macaddr	Unicast MAC address for which the device has forwarding information.

Output Parameters

ret	Pointer to the port number.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_next_rbridge_unifdb_nick

This call gets the RBridge nickname which is placed in the Egress Nickname field of a TRILL frame for the next unicast mac address.

Syntax

```
int
trill_get_next_rbridge_unifdb_nick (u_int32_t vr_id, u_int32_t *instance,
                                    u_int32_t *fdbid, u_char **macaddr,
                                    int indexlen, u_int32_t  *ret)
```


Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`fdbid`	Identity of this filtering database.
`macaddr`	Unicast MAC address for which the device has forwarding information.
`indexlen`	Index length.

Output Parameters

`ret`	RBridge nickname.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_next_rbridge_unifdb_confidence

This call gets the confidence level associated with next unicast FDB entry.

Syntax

```
int
trill_get_next_rbridge_unifdb_confidence (u_int32_t vr_id, u_int32_t *instance,
                                          u_int32_t *fdbid, u_char **macaddr,
                                          int indexlen, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`fdbid`	Identity of this filtering database.
`macaddr`	Unicast MAC address for which the device has forwarding information.
`indexlen`	Index length.

Output Parameters

`ret`	Confidence level.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_next_rbridge_unifdb_status

This call gets status of next unicast FDB entry.

Syntax

```
int
trill_get_next_rbridge_unifdb_status (u_int32_t vr_id, u_int32_t *instance,
```

```
                                    u_int32_t *fdbid, u_char **macaddr,
                                    int indexlen, u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
fdbid	Identity of this filtering database.
macaddr	Unicast MAC address for which the device has forwarding information.
indexlen	Index length.

Output Parameters

ret	Status of UniFdb.
1	Other
2	Invalid
3	Learned
4	Self
5	mgmt
6	ESADI

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, DOT1QFDBID and MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, DOT1QFDBID and MacAddr not found.

trill_get_rbridge_unifib_macaddress

This call gets the MAC address of the next-hop RBridge for the path towards the RBridge, whose nickname is specified in this entry.

Syntax

```
int
trill_get_rbridge_unifib_macaddress (u_int32_t vr_id, u_int32_t instance,
                                     u_int16_t nickname, u_int32_t port,
                                     u_char **ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
nickname	An RBridge nickname for which this RBridge has forwarding information.
port	Port number of the port attached to the next-hop bridge.

Output Parameters

ret	MAC address of the next-hop RBridge for the path towards the specified RBridge.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, Nickname and Port.

TRILL_API_GET_ERROR for valid Instance ID, Nickname and Port not found.

trill_get_rbridge_unifib_macaddress

This call gets the MAC address of the next-hop RBridge for the path towards the RBridge, whose nickname is specified by the next entry.

Syntax

```
int
trill_get_rbridge_unifib_macaddress (u_int32_t vr_id, u_int32_t instance,
                                     u_int16_t nickname, u_int32_t port,
                                     u_char **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	An RBridge nickname for which this RBridge has forwarding information.
`port`	Port number of the port attached to the next-hop bridge.
`indexlen`	An Integer variable that contains the length of the index

Output Parameters

`ret`	MAC address of the next-hop RBridge for the path towards the next RBridge.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, Nickname and Port.

TRILL_API_GET_ERROR for valid Instance ID, Nickname and Port not found.

trill_get_rbridge_multifib_port

This call gets the list of ports to which a frame destined to this multicast distribution tree is flooded

Syntax

```
int
trill_get_rbridge_multifib_port (u_int32_t vr_id, u_int32_t instance,
                                 u_int16_t nickname, void  *portlist)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname of the multicast distribution tree.

Output Parameters

`portlist`	An octet string containing the list of ports.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, Nickname.

TRILL_API_GET_ERROR for valid Instance ID, Nickname not found.

trill_get_next_rbridge_multifib_port

This call gets the list of ports to which a frame destined to the next multicast distribution tree is flooded

Syntax:

```
int
trill_get_next_rbridge_multifib_port (u_int32_t vr_id, u_int32_t *instance,
                                      u_int16_t *nickname, int indexlen,
                                      void  *portlist)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nickname`	Nickname of the multicast distribution tree.
`indexlen`	An Integer variable that contains the length of the index

Output Parameters

`portlist`	An octet string containing the list of ports.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, Nickname.

TRILL_API_GET_ERROR for valid Instance ID, Nickname not found.

trill_get_rbridge_vlan_forwarderlost

This call gets the number of times this RBridge has lost appointed forwarder status for this VLAN on any of its ports.

Syntax

```
int
trill_get_rbridge_vlan_forwarderlost (u_int32_t vr_id, u_int32_t instance,
                                      u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`ret`	Number of times the RBridge has lost AF status.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_get_rbridge_vlan_disablelearning

This call gets disable learning value of MAC addresses seen in this VLAN.

Syntax

```
int
trill_get_rbridge_vlan_disablelearning (u_int32_t vr_id, u_int32_t instance,
                                        u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`ret`	True or False value of disable learning.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_get_rbridge_vlan_snooping

This call gets IP Multicast Snooping on this VLAN.

Syntax

```
int
trill_get_rbridge_vlan_snooping (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t vlanindex, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

Output Parameters

`ret`	An integer value containing either: notSupported (1) ,ipv4 (2) ,ipv4v6 (3).

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_get_next_rbridge_forwarderlost

This call gets the number of times the next RBridge has lost appointed forwarder status for this VLAN on any of its ports.

Syntax

```
int
trill_get_next_rbridge_vlan_forwarderlost (u_int32_t vr_id, u_int32_t *instance,
                                           u_int32_t *vlanindex, int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`indexlen`	An Integer variable that contains the length of the index

Output Parameters

`ret`	Number of times the next RBridge has lost AF status.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_get_next_rbridge_disablelearning

This call gets disable learning value of MAC addresses seen in the next VLAN.

Syntax

```
int
trill_get_next_rbridge_vlan_disablelearning (u_int32_t vr_id,
                                             u_int32_t *instance,
                                             u_int32_t *vlanindex,
                                             int indexlen,
                                             u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`indexlen`	An Integer variable that contains the length of the index

Output Parameters

`ret`	True or false value of disable learning.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_get_next_rbridge_vlan_snooping

This call gets IP Multicast Snooping on the next VLAN.

Syntax

```
int
trill_get_next_rbridge_vlan_snooping (u_int32_t vr_id, u_int32_t *instance,
                                      u_int32_t *vlanindex, int indexlen,
                                      u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`indexlen`	An Integer variable that contains the length of the index

Output Parameters

`ret`	An integer value containing either: notSupported (1) ,ipv4 (2) ,ipv4v6 (3)

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index not found.

trill_set_rbridge_vlan_disablelearning

This call sets disable learning value of MAC addresses seen in this VLAN.

Syntax

```
int
trill_set_rbridge_vlan_disablelearning (u_int32_t vr_id, u_int32_t instance,
                                        u_int32_t vlanindex, u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`val`	True or False value of disable learning.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID, VLAN index.

TRILL_API_SET_ERROR for valid Instance ID, VLAN index not found.

trill_get_rbridge_vlanport_inhibited

This call gets whether this VLAN has been inhibited by the RBridge due to conflicting Forwarder information received from another RBridge.

Syntax

```
int
trill_get_rbridge_vlanport_inhibited (u_int32_t vr_id, u_int32_t instance,
                                      u_int16_t vlanindex, u_int32_t baseport,
                                      u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.

Output Parameters

`ret`	True or false value of VlanPort_Inhibited.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_rbridge_vlanport_forwarder

This call gets whether this RBridge is an Appointed Forwarder for this VLAN on this port.

Syntax

```
int
trill_get_rbridge_vlanport_forwarder (u_int32_t vr_id, u_int32_t instance,
                                      u_int16_t vlanindex, u_int32_t baseport,
                                      u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.

Output Parameters

ret	True or false value of VlanPort_Forwarder.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_rbridge_vlanport_announcing

This call gets whether TRILL-Hellos tagged with this VLAN can be sent by this RBridge on this port.

Syntax

```
int
trill_get_rbridge_vlanport_announcing (u_int32_t vr_id, u_int32_t instance,
                                       u_int16_t vlanindex, u_int32_t baseport,
                                       u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.
vlanindex	An integer containing the VLAN index.
baseport	An Integer containing the port number of the port for which this entry contains RBridge management information.

Output Parameters

ret	True or false value of VlanPort_Announcing.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_rbridge_vlanport_detectedvlanmapping

This call gets whether VLAN mapping has been detected on the link attached to this port.

Syntax

```
int
trill_get_rbridge_vlanport_detectedvlanmapping (u_int32_t vr_id,
                                                u_int32_t instance,
                                                u_int16_t vlanindex,
                                                u_int32_t baseport,
                                                u_int32_t  *ret)
```

Input Parameters

vr_id	An integer that contains the TRILL VR identifier.
instance	An integer that contains the TRILL instance ID.

`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.

Output Parameters

`ret`	True or false value of VlanPort_DetectedVlanMapping.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_next_rbridge_vlanport_inhibited

This call gets whether the next VLAN has been inhibited by the RBridge due to conflicting Forwarder information received from another RBridge.

Syntax

```
int
trill_get_next_rbridge_vlanport_inhibited (u_int32_t vr_id,
                                           u_int32_t *instance,
                                           u_int16_t *vlanindex,
                                           u_int32_t *baseport,
                                           int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	True or false value of VlanPort_Inhibited.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_next_rbridge_vlanport_forwarder

This call gets whether the next RBridge is an Appointed Forwarder for this VLAN on this port.

Syntax

```
int
```

```
trill_get_next_rbridge_vlanport_forwarder (u_int32_t vr_id,
                                           u_int32_t *instance,
                                           u_int16_t *vlanindex,
                                           u_int32_t *baseport,
                                           int indexlen,
                                           u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	True or false value of VlanPort_Forwarder.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_next_rbridge_vlanport_announcing

This call gets whether TRILL-Hellos tagged with the next VLAN can be sent by this RBridge on this port.

Syntax

```
int
trill_get_next_rbridge_vlanport_announcing (u_int32_t vr_id,
                                            u_int32_t *instance,
                                            u_int16_t *vlanindex,
                                            u_int32_t *baseport,
                                            int indexlen,
                                            u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	True or false value of VlanPort_Announcing.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_next_rbridge_vlanport_detectedvlanmapping

This call gets whether VLAN mapping has been detected on the link attached to the next port.

Syntax

```
int
trill_get_next_rbridge_vlanport_detectedvlanmapping (u_int32_t vr_id,
                                                     u_int32_t *instance,
                                                     u_int16_t *vlanindex,
                                                     u_int32_t *baseport,
                                                     int indexlen,
                                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	True or false value of VlanPort_DetectedVlanMapping.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_set_rbridge_vlanport_announcing

This call sets whether TRILL-Hellos tagged with this VLAN can be sent by this RBridge on this port.

Syntax

```
int
trill_set_rbridge_vlanport_announcing (u_int32_t vr_id, u_int32_t instance,
                                       u_int16_t vlanindex, u_int32_t baseport,
                                       u_int32_t  val)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An integer containing the VLAN index.

`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`val`	True or false value of VlanPort_Announcing to be set.

Output Parameters

None

Return Value

TRILL_API_SET_SUCCESS for valid Instance ID, VLAN index and BasePort.

TRILL_API_SET_ERROR for valid Instance ID, VLAN index and BasePort not found.

trill_get_rbridge_snoopingport_addrtype

This call gets the IP address type of an IP multicast router detected on this port.

Syntax

```
int
trill_get_rbridge_snoopingport_addrtype (u_int32_t vr_id, u_int32_t instance,
                                         u_int32_t baseport, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`instance`	An integer that contains the Trill instance ID.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`val`	True or false value of VlanPort_Announcing to be set.

Output Parameters

None

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, BasePort.

TRILL_API_GET_ERROR for valid Instance ID, BasePort not found.

trill_get_rbridge_snoopingport_addr

This call gets the IP address of an IP multicast router detected on this port.

Syntax

```
int
trill_get_rbridge_snoopingport_addr (u_int32_t vr_id, u_int32_t instance,
                                     u_int32_t baseport, u_char  **ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.

`instance`	An integer that contains the TRILL instance ID.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.

Output Parameters

`ret`	An octet string containing the IP address.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, BasePort.

TRILL_API_GET_ERROR for valid Instance ID, BasePort not found.

trill_get_next_rbridge_snoopingport_addrtype

This call gets the IP address type of an IP multicast router detected on the next port.

Syntax

```
int
trill_get_next_rbridge_snoopingport_addrtype (u_int32_t vr_id,
                                              u_int32_t *instance,
                                              u_int32_t *baseport,
                                              int indexlen,
                                              u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing the IP address type.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, BasePort.

TRILL_API_GET_ERROR for valid Instance ID, BasePort not found.

trill_get_next_rbridge_snoopingport_addr

This call gets the IP address of an IP multicast router detected on the next port.

Syntax:

```
int
trill_get_next_rbridge_snoopingport_addrtype (u_int32_t vr_id,
                                              u_int32_t *instance,
                                              u_int32_t *baseport,
                                              int indexlen,
```



```
                                                u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`baseport`	An Integer containing the port number of the port for which this entry contains RBridge management information.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An octet string containing the IP address.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, BasePort.

TRILL_API_GET_ERROR for valid Instance ID, BasePort not found.

trill_get_next_rbridge_snooping_addrports

This call gets the set of ports on which a listener has been detected for this IP multicast address.

Syntax

```
int
trill_get_next_rbridge_snooping_addrports (u_int32_t vr_id,
                                           u_int32_t *instance,
                                           u_int32_t *vlanindex,
                                           u_int32_t *addrtype,
                                           struct prefix *prefix,
                                           int indexlen,
                                           void  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An Integer containing the VLAN index
`addrtype`	An Integer containing th IP Address type.
`prefix`	IP Address

Output Parameters

`ret`	An octet string containing the list of ports.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index, AddrType and Addr.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index, AddrType and Addr not found.

trill_get_next_rbridge_snooping_addrports

This call gets the set of ports on which a listener has been detected for the next IP multicast address.

Syntax

```
int
trill_get_next_rbridge_snooping_addrports (u_int32_t vr_id,
                                           u_int32_t *instance,
                                           u_int32_t *vlanindex,
                                           u_int32_t *addrtype,
                                           struct prefix *prefix,
                                           int indexlen,
                                           void  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`vlanindex`	An Integer containing the VLAN index
`addrtype`	An Integer containing th IP Address type.
`prefix`	IP Address
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An octet string containing the list of ports.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, VLAN index, AddrType and Addr.

TRILL_API_GET_ERROR for valid Instance ID, VLAN index, AddrType and Addr not found.

trill_get_rbridge_dtree_nick

This call gets the nickname of the distribution tree.

Syntax

```
int
trill_get_rbridge_dtree_nick (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t nicknumber, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nicknumber`	An Integer containing the Dtree Number.

Output Parameters

`ret`	An integer containing the Nickname of the Dtree.

Return Value:

TRILL_API_GET_SUCCESS for valid Instance ID, NickNumber.

TRILL_API_GET_ERROR for valid Instance ID, NickNumber not found.

trill_get_rbridge_dtree_ingress

This call gets whether this RBridge might choose this distribution tree to ingress a multi-destination frame.

Syntax:

```
int
trill_get_rbridge_dtree_ingress (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t nicknumber, u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nicknumber`	An Integer containing the Dtree Number.

Output Parameters

`ret`	An integer containing True (1) or False (0) value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, NickNumber.

TRILL_API_GET_ERROR for valid Instance ID, NickNumber not found.

trill_get_next_rbridge_dtree_nick

This call gets the nickname of the next entry of the distribution tree.

Syntax

```
int
trill_get_next_rbridge_dtree_nick (u_int32_t vr_id, u_int32_t *instance,
                                   u_int32_t *nicknumber, int indexlen,
                                   u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nicknumber`	An Integer containing the Dtree Number.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing the Nickname of the Dtree.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, NickNumber.

TRILL_API_GET_ERROR for valid Instance ID, NickNumber not found.

trill_get_next_rbridge_dtree_ingress

This call gets whether this RBridge might choose the next entry of distribution tree to ingress a multi-destination frame.

Syntax

```
int
trill_get_next_rbridge_dtree_ingress (u_int32_t vr_id, u_int32_t *instance,
                                      u_int32_t *nicknumber, int indexlen,
                                      u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`nicknumber`	An Integer containing the Dtree Number.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing True (1) or False (0) value.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, NickNumber.

TRILL_API_GET_ERROR for valid Instance ID, NickNumber not found.

trill_get_next_rbridge_trillnbr_mtu

This call gets the MTU size to this neighbor for IS-IS communication purposes.

Syntax

```
int
trill_get_next_rbridge_trillnbr_mtu (u_int32_t vr_id,
                                     u_int32_t *instance,
                                     struct trill_mac_addr *macaddr,
                                     int indexlen,
                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`macaddr`	An octet string containing the MAC Address.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing MTU size.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, MacAddr not found.

trill_get_rbridge_trillnbr_failedmtutest

This call gets whether the neighbor's tested MTU is less than the minimum acceptable inter-bridge link MTU for the campus (1470).

Syntax

```
int
trill_get_rbridge_trillnbr_failedmtutest (u_int32_t vr_id,
                                          u_int32_t instance,
                                          struct trill_mac_addr macaddr,
                                          u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`macaddr`	An octet string containing the MAC Address.

Output Parameters

`ret`	An integer containing either True (1) or False (0).

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, MacAddr not found.

trill_get_next_rbridge_trillnbr_mtu

This call gets the MTU size to the next entry of neighbor for IS-IS communication purposes.

Syntax

```
int
trill_get_next_rbridge_trillnbr_mtu (u_int32_t vr_id,
                                     u_int32_t *instance,
                                     struct trill_mac_addr *macaddr,
                                     int indexlen,
                                     u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`macaddr`	An octet string containing the MAC Address.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing MTU size.

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, MacAddr not found.

trill_get_next_rbridge_trillnbr_failedmtutest

This call gets whether the next neighbor's tested MTU is less than the minimum acceptable inter-bridge link MTU for the campus (1470).

Syntax

```
int
trill_get_next_rbridge_trillnbr_failedmtutest (u_int32_t vr_id,
                                               u_int32_t *instance,
                                               struct trill_mac_addr *macaddr,
                                               int indexlen,
                                               u_int32_t  *ret)
```

Input Parameters

`vr_id`	An integer that contains the TRILL VR identifier.
`instance`	An integer that contains the TRILL instance ID.
`macaddr`	An octet string containing the MAC Address.
`indexlen`	An integer containing the length of the index.

Output Parameters

`ret`	An integer containing either True (1) or False (0).

Return Value

TRILL_API_GET_SUCCESS for valid Instance ID, MacAddr.

TRILL_API_GET_ERROR for valid Instance ID, MacAddr not found.

CHAPTER 15 Acronyms

Term	Definition
AF	Appointed Forwarder
BPDU	Bridge PDU
CHbH	Critical Hop-by-Hop
CItE	Critical Ingress-to-Egress
CLI	Command Line Interface
DMAC	Destination MAC (address)
DR	Designated Router
DRB	Designated Router Bridge
ECMP	Equal Cost Multi-Path
FDB	Forwarding Database
GARP	Generic Attribute Registration Protocol
GVRP	GARP VLAN Registration Protocol
HAL	Hardware Abstraction Layer
IGMP	Internet Group Management Protocol
IMI	Integrated Management Interface
IS-IS	Intermediate System to Intermediate System
LSP	Link State PDU
MAC	Media Access Control
MLD	Multicast Listener Discovery
MRD	Multicast Router Discovery
MTU	Maximum Transmission Unit
NSM	Network Services Module
P2P	Point-to-point
PDU	Protocol Data Unit
RBridge	Routing Bridge

Term	Definition
RPF	Reverse Path Forwarding
SNMP	Simple Network Management Protocol
SPF	Shortest Path First
STP	Spanning Tree Protocol
TLV	Type
TRILL	Transparent Interconnection of Lots of Links
VLAN	Virtual Local Area Network

Index

L

M

N

P

R

S

T

U

V