ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Intermediate System to Intermediate System
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 Intermediate System to Intermediate System (IS-IS).

Audience

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

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:

Related Documents

The following guides are related to this document:

- *Intermediate System to Intermediate System Command Reference*
- *Installation Guide*
- *Network Services Module Developer Guide*
- *Network Services Module Command Reference*
- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 Introduction

The Intermediate System-to-Intermediate System (IS-IS) protocol is a two-level hierarchical interior gateway protocol (IGP) for routing both IP and OSI, using a link-state in the individual areas that make up the hierarchy. The Shortest Past First (SPF) computation is used to calculate the shortest path tree (SPT) inside each area.

A 2-level hierarchy is used to support large routing domains. A large domain may be administratively divided into areas. Each system resides in exactly one area. Routing within an area is called Level-1 routing. Routing between areas is called Level-2 routing. A Level-2 Intermediate System (IS) keeps track of the paths to destination areas. A Level-1 IS keeps track of the routing within its own area. For a packet destined for another area, a Level-1 IS sends the packet to the nearest Level-2 IS in its own area, regardless of the destination area. Then, the packet travels, via Level-2 routing, to the destination area, where it may travel, via Level-1 routing, to the destination.

Note: Selecting an exit from an area based on Level-1 routing to the closest Level-2 IS might result in suboptimal routing.

SPF Hold Time

The Link State database consists of information from the Link State Protocol Data Units (LSPDU) from every other IS. The SPF algorithm gleans information from these LSPs to prepare the SPT. The SPF algorithm is calculated every time a change occurs in the topology within the area. Frequent SPF calculations for each change can affect other processes running on the same CPU. Instead of doing the SPF calculation after a fixed delay (called the hold-time), when a change in topology is received via the LSPs, the hold- time is made variable and configurable.

IS-IS uses an exponential back-off algorithm for hold-time delay calculation and has a configurable hold-time. This provides improved scalability to accommodate frequent topology changes. It also provides faster convergence when topology changes are at a slower rate.

Hold-Time Calculation Algorithm

The calculation of the hold-time uses an exponential mechanism. The CurrHold value designates the current hold-time value to use for delaying SPF calculation. The CurrHold time is initialized to the value of the minimum hold-time (MinHold) at the start of processing. If a topology change notification is received before the CurrHold time, the SPF calculation is delayed by either the MinHold time or the CurrHold time since the last SPF calculation, whichever is less. The CurrHold time is set to either the SPF_INCREMENTAL_VALUE of its current value, or to the MaxHold value, whichever is less.

If a topology change notification is received after the CurrHold time into the SPF_INCREMENTAL_VALUE, the CurrHold is set to the value of the MinHold time. In this case, SPF calculation is delayed by the MinHold time or CurrHold time since the last SPF, whichever is less.

Setting Hold Time

Maximum and minimum hold-time intervals between SPF calculations are set using the `spf-interval-exp` command. See the *Intermediate System to Intermediate System Command Reference* for details.

Authentication

The IS-IS protocol, as specified in ISO 10589 [1], provides for LSP authentication with the inclusion of authentication information as part of the LSP. This authentication information is encoded as a Type-Length-Value (TLV) tuple. The TLV type is specified as 10. The TLV length is variable. The value of the TLV depends on the authentication algorithm and related confidential information used. The first octet of the value specifies the authentication type. Type 0 is reserved; type 1 indicates a clear text password, and type 255 is used for routing domain private authentication methods. The remainder of the TLV value is the authentication value.

IS-IS HMAC-MD5 Authentication

IS-IS HMAC-MD5 authentication provides more security using the HMAC: Keyed-Hashing for Message Authentication, as defined in RFC 2104. The authentication type used for HMAC-MD5 is 54(0x36).The length of the message digest for HMAC-MD5 is 16, and the length field in the TLV is 17. IS-IS HMAC-MD5 authentication adds an HMAC-MD5 digest to each IS-IS protocol data unit (PDU). HMAC is a mechanism for message authentication codes (MACs) using cryptographic hash functions. The digest allows authentication at the IS-IS routing protocol level, which prevents unauthorized routing message from being injected into the network routing domain.

IS-IS has five PDU types: LSP, LAN Hello, Point-to-point Hello, CSNP, and PSNP. IS-IS HMAC-MD5 authentication can be applied to all five types of PDUs. Authentication can be independently enabled on different IS-IS levels. The interface-related PDUs (LAN Hello, Point-to-point Hello, CSNP, and PSNP) can be enabled with authentication on different interfaces, with different levels and different passwords.

Setting Authentication

Authentication can be set at the interface or instance level. See the *Intermediate System to Intermediate System Command Reference* for details.

IS-IS and OSPF Commonalities

IS-IS and OSPF have several characteristics in common:

- Link state update
- Link state database
- Hello for adjacency
- Classless protocol
- Areas for hierarchical topology
- IPv4 and IPv6 routing

CHAPTER 2 Data Structures and Databases

This chapter describes the data structures and databases of the IS-IS protocol module.

Data Structures

The following subsection describes the data structures for the IS-IS protocol module.

Common Data Structures

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

- `pal_in4_addr`
- `pal_in6_addr`
- `prefix`

isis_lspid

This data structure contains LSP-related information and flags.

Definition

```
struct isis_lspid
{
  u_char source_id[ISIS_SYSID_LENGTH];
  u_char psn_id;
  u_char lsp_num;
};
```

isis

This data structure contains both level-1 and level-2 sub-structures. It has the interface and global neighbor tables, and also contains system-wide configuration, including system ID, area addresses, and redistribute source information.

Definition

```
struct isis
{
  /* Tag of IS-IS area.  */
  char *tag;

  /* Instance ID.  */
  u_int32_t instance_id;

  /* Pointer to IS-IS master.  */
  struct isis_master *im;

  /* VRF binding. */
```

```
  struct isis_vrf *iv;

  /* IS-IS start time.  */
  pal_time_t start_time;

  /* IS-IS System ID.  */
  u_char system_id[ISIS_SYSID_LENGTH];

  /* IS-type.  */
  u_char is_type;

  /* IS-IS protocols supported.  */
  u_char proto_type;

  /* Priority tag */
  u_int32_t priority_tag;

 /* IS-IS administrative flags.  */
  u_char flags;
#define ISIS_FLAG_OVERLOAD                      (1 << 0)
#define ISIS_FLAG_SHUTDOWN                      (1 << 1)

  /* Config flags.  */
  u_int16_t config;
#define ISIS_CONFIG_SYSTEM_ID                   (1 << 0)
#define ISIS_CONFIG_LSP_MTU                     (1 << 1)
#define ISIS_CONFIG_LSP_REFRESH_INTERVAL        (1 << 2)
#define ISIS_CONFIG_MAX_LSP_LIFETIME            (1 << 3)
#define ISIS_CONFIG_IGNORE_LSP_ERRORS           (1 << 4)
#define ISIS_CONFIG_DYNAMIC_HOSTNAME            (1 << 5)
#define ISIS_CONFIG_HOSTNAME_AREA_TAG           (1 << 6)
#define ISIS_CONFIG_TE_ROUTER_ID                (1 << 7)
#define ISIS_CONFIG_SET_OVERLOAD_BIT            (1 << 8)
#ifdef HAVE_BFD
#define ISIS_CONFIG_BFD                         (1 << 9)
#endif /* HAVE_BFD */
#define ISIS_CONFIG_PASSIVE_INTERFACES          (1 << 10)
#ifdef HAVE_WIDE_METRIC
#define ISIS_CONFIG_HIGH_PRIORITY_TAG           (1 << 11)
#endif /* HAVE_WIDE_METRIC */
#ifdef HAVE_ISIS_CSPF
#define ISIS_CONFIG_CSPF                        (1 << 12)
#endif /* HAVE_ISIS_CSPF */

/* Config variables.  */
  u_char max_area_addrs;                /* Number of Max Area Addresses. */
  vector area_addrs;                    /* Manual Area Addresses. */
  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. */
#ifdef HAVE_ISIS_TE
  struct pal_in4_addr router_id;        /* TE router-ID. */
#endif /* HAVE_ISIS_TE */

  /* Overload bit flags.  */
```



```
  u_char overload_flags;
#define ISIS_OL_BIT_ON_STARTUP          (1 << 0)
#define ISIS_OL_BIT_SUPPRESS_EXTERNAL   (1 << 1)
#define ISIS_OL_BIT_SUPPRESS_INTERLEVEL (1 << 2)
#define ISIS_OL_BIT_WAIT_FOR_BGP        (1 << 3)

  /* Overload timer flags to capture the overload state of the system. */
  u_char overload_timer_flags;
  /* Overload timer is running. */
#define ISIS_OL_TIMER_ON         (1 << 0)
  /* Overload timer has to be started. */
#define ISIS_OL_TIMER_START      (1 << 1)
  /* Overload bit has to be cleared in the LSPs. */
#define ISIS_OL_TIMER_CLEAR_ALL  (1 << 2)

  /* Overload interval the timer is started with. */
  u_int32_t overload_started_val;

u_int32_t overload_interval;          /* Overload interval after reboot. */
  struct pal_timeval tv_overload;       /* Overload interval start time. */

  /* Passive interfaces */
  struct list *passive_if;       /* List of passive interfaces */
  struct list *no_passive_if;    /* List of no passive interfaces */
  bool_t passive_if_default;     /* To check passive interface is set or not
*/

  /* Information per protocol.  */
  struct isis_proto proto[ISIS_PROTO_INDEX_MAX];

  /* Level context.  */
  struct isis_level level[ISIS_LEVEL_INDEX_MAX];

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

  /* Distance parameter */
  struct ls_table *distance_table;
  u_char distance [ISIS_PROTO_INDEX_MAX];

  /* Tables.  */
  struct ls_table *if_table;            /* IS-IS inteface table. */
  struct ls_table *nbr_table;           /* Global neighbor table. */
  struct ls_table *nexthop_table;       /* Nexthop table. */

 /* TLVs.  */
  struct isis_tlv *tlv_area_addrs;      /* Area Addresses. */
  struct isis_tlv *tlv_protos;          /* Protocol supported. */
  struct isis_tlv *tlv_hostname;        /* Hostname. */
#ifdef HAVE_MULTI_TOPOLOGY
  struct isis_tlv *tlv_multi;           /* Multi-topology TLV. */
#endif /* HAVE_MULTI_TOPOLOGY */

  /* Threads.  */
  struct thread *t_maxage_walker;       /* MaxAge walker. */
  struct thread *t_overload;            /* Overload on-startup timer. */
```

```
#ifdef HAVE_ISIS_CSPF
  /* CSPF. */
  struct cspf *cspf;
#endif /* HAVE_ISIS_CSPF */
};
```

isis_nbr_level

This data structure contains IS-IS neighbor state, holdtime, and priority for each level.

Definition

```
struct isis_nbr_level
{
  /* Pointer to parent neighbor. */
  struct isis_neighbor *nbr;

  /* Pointer to the parent interface level. */
  struct isis_if_level *ifl;

  /* Neighbor index. */
  u_char index;

  /* Neighbor adjacency ID.  */
  u_int32_t adjacency_id;

  /* NSM state. */
  u_char state;

  /* Priority. */
  u_char priority;

  /* Supported protocols flags. */
  u_char proto;

 /* DIS ID. */
  struct isis_dis_id lan_id;

  /* IS-reachability information for PSN-LSP. */
  struct isis_is_reach_info *is_reach;

  /* Holding time. */
  u_int16_t hold_time;

  /* Thread. */
  struct thread *t_hold_timer;
  struct thread *t_events[ISIS_LAN_NSM_EVENT_MAX];
};
```

isis_if_level

This data structure contains level-specific information related to the IS-IS interface. Contains PSNP send queue, the DIS and the circuit ID for broadcast networks.

Definition

```
struct isis_if_level
{
  /* Level index. */
  u_char index;

  /* Level flags. */
  u_char flags;
#define ISIS_IF_LEVEL_DIS                (1 << 0)
#define ISIS_IF_LEVEL_RR                 (1 << 1)
#define ISIS_IF_LEVEL_RA                 (1 << 2)
#ifdef HAVE_RESTART
#define ISIS_IF_LEVEL_SA                 (1 << 3)
#endif /* HAVE_RESTART */

  /* ISM State. */
  u_char state;
  u_char ostate;

  /* Pointer to parent interface. */
  struct isis_interface *isi;

  /* Pointer to level context of IS-IS instance. */
  struct isis_level *il;

  /* DIS of this interface. */
  struct isis_dis_id dis;

  /* IS-reachability information. */
  struct isis_is_reach_info *is_reach;          /* for REG-LSP. */
  struct isis_is_reach_info *is_reach_psn;      /* for PSN-LSP. */

 /* Vector of IS-neighbor map for PSN-LSP. */
  vector is_map_vec;

  /* PSNP list. */
  struct list *psnp;

  /* TLVs. */
  struct isis_tlv *tlv_auth_info;       /* Authentication Info. */
#ifdef HAVE_MD5
  struct isis_tlv *tlv_hmac_md5_auth_info;  /* MD5 Authentication Info.*/
#endif /*HAVE_MD5 */

#ifdef HAVE_GMPLS
  struct isis_tlv *tlv_srlg;            /* Shared Risk Link Group. */
#endif /* HAVE_GMPLS */

  /* Threads. */
  struct thread *t_hello;               /* Hello timer. */
  struct thread *t_psnp;                /* PSNP timer. */
  struct thread *t_csnp;                /* CSNP timer for DIS. */
  struct thread *t_dis_election;        /* DIS election. */
  struct thread *t_is_reach_map;        /* IS-reachability timer. */
#ifdef HAVE_RESTART
  struct thread *t_restart_dis;         /* Restart helper DIS election. */
```



```
#endif /* HAVE_RESTART */

 /* Statistics. */
  u_int32_t adj_changes;                 /* isisCircAdjChanges. */
  u_int32_t num_adj;                     /* isisCircNumAdj. */
  u_int32_t init_fails;                  /* isisCircInitFails. */
  u_int32_t rej_adjs;                    /* isisCircRejAdjs. */
  u_int32_t id_len_mismatches;           /* isisCircIDFieldLenMismatches. */
  u_int32_t max_area_addr_mismatches;    /* isisCircMaxAreaAddrMismatches. */
  u_int32_t auth_type_fails;             /* isisCircAuthTypeFails. */
  u_int32_t auth_fails;                  /* isisCircAuthFails. */
  u_int32_t dis_changes;                 /* isisCircLANDesISChanges */

  u_int32_t state_change;                /* Number of ISM state change. */

  u_int32_t hello_in;                    /* isisPacketCountIIHello. */
  u_int32_t hello_out;                   /* isisPacketCountIIHello. */
  u_int32_t lsp_in;                      /* isisPacketCountLSP. */
  u_int32_t lsp_out;                     /* isisPacketCountLSP. */
  u_int32_t csnp_in;                     /* isisPacketCountCSNP. */
  u_int32_t csnp_out;                    /* isisPacketCountCSNP. */
  u_int32_t psnp_in;                     /* isisPacketCountPSNP. */
  u_int32_t psnp_out;                    /* isisPacketCountPSNP. */
  u_int32_t unknown_in;                  /* isisPacketCountUnknown. */
  u_int32_t unknown_out;                 /* isisPacketCountUnknown. */

#ifdef HAVE_RESTART
  u_int8_t  t1_exp;                      /* number of times T1 timer expired.
*/
#endif /* HAVE_RESTART */
};
```

isis_packet

This data structure contains IS-IS packet-related information, and a pointer to the RAW packet data stream.

Definition

```
struct isis_packet
{
  /* Pointer of next packet. */
  struct isis_packet *next;

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

  /* TLV vector. */
  vector tlvvec;

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

  /* Level Index. */
  u_char index;
};
```

isis_level

This data structure contains IS-IS level specific information including LSPDB table, SPF tree, IP routing table, and level-specific configurations.

Definition

```
struct isis_level
{
  /* Pointer to IS-IS instance. */
  struct isis *top;

  /* Index. */
  u_char index;

  /* Flags. */
  u_char flags;
#define ISIS_LEVEL_UP                           (1 << 0)
#define ISIS_LEVEL_WAIT_MAXSEQNUM               (1 << 1)
#define ISIS_LEVEL_RESTARTING                   (1 << 2)
#define ISIS_LEVEL_SYNC_FAIL                    (1 << 3)
#define ISIS_LEVEL_RESTART_OVERLOAD             (1 << 4)
#ifdef HAVE_RESTART
#define ISIS_LEVEL_STARTING                     (1 << 5)
#define ISIS_LEVEL_SA                           (1 << 6)
#endif /* HAVE_RESTART */

   /* Configuration flags. */
  u_int16_t config;
#define ISIS_CONFIG_LSP_GEN_INTERVAL            (1 << 0)
#define ISIS_CONFIG_SPF_INTERVAL                (1 << 1)
#define ISIS_CONFIG_ORG_LSP_BUFSIZE             (1 << 2)
#define ISIS_CONFIG_LEVEL_PASSWD                (1 << 3)
#define ISIS_CONFIG_TE_ENABLED                  (1 << 4)
#define ISIS_CONFIG_RESTART_TIMER               (1 << 5)
#define ISIS_CONFIG_SET_SNP_AUTH_VALIDATE       (1 << 6)
#define ISIS_CONFIG_SET_SNP_AUTH_SEND_ONLY      (1 << 7)
#define ISIS_CONFIG_SET_AUTH_SEND_ONLY          (1 << 8)
#define ISIS_CONFIG_SET_AUTH_MODE_MD5           (1 << 9)
#define ISIS_CONFIG_SET_AUTH_MODE_TEXT          (1 << 10)
#define ISIS_CONFIG_SET_AUTH_KEY_CHAIN          (1 << 11)
#define ISIS_CONFIG_ISPF                        (1 << 12)
#define ISIS_CONFIG_PRC                         (1 << 13)

  /* Level Configuration variables. */

  /* SPF timer config. */
  struct pal_timeval spf_min_delay;         /* SPF minimum delay time. */
  struct pal_timeval spf_max_delay;         /* SPF maximum delay time. */

  /* PRC timer config. */
  struct pal_timeval prc_min_delay;         /* PRC minimum delay time. */
  struct pal_timeval prc_max_delay;         /* PRC maximum delay time. */
#define ISIS_SPF_INCREMENT_VALUE        5

 u_char lsp_gen_interval;              /* LSP Minimum gen interval. */
  u_int16_t lsp_bufsize;                /* LSP orig bufsize. */
```

```
  char *passwd;                          /* Area/Domain passwd. */
#define area_passwd     level[L1_INDEX].passwd
#define domain_passwd   level[L2_INDEX].passwd
#ifdef HAVE_RESTART
  u_int16_t restart_timer;               /* Restart sync timer. */
#endif /* HAVE_RESTART */

  /* Authentication key-chain. */
  char *key_chain;

  /* Metric style. */
  u_char metric_style;
#define ISIS_METRIC_TLV_NARROW                   (1 << 0)
#define ISIS_METRIC_SPF_NARROW                   (1 << 1)
#define ISIS_METRIC_TLV_WIDE                     (1 << 2)
#define ISIS_METRIC_SPF_WIDE                     (1 << 3)

#define ISIS_METRIC_NARROW                   \
    (ISIS_METRIC_TLV_NARROW|ISIS_METRIC_SPF_NARROW)
#define ISIS_METRIC_NARROW_TRANSITION                        \
    (ISIS_METRIC_TLV_NARROW|ISIS_METRIC_SPF_NARROW|ISIS_METRIC_SPF_WIDE)
#define ISIS_METRIC_WIDE                             \
    (ISIS_METRIC_TLV_WIDE|ISIS_METRIC_SPF_WIDE)
#define ISIS_METRIC_WIDE_TRANSITION                         \
    (ISIS_METRIC_TLV_WIDE|ISIS_METRIC_SPF_NARROW|ISIS_METRIC_SPF_WIDE)
#define ISIS_METRIC_TRANSITION                                 \
    (ISIS_METRIC_TLV_NARROW|ISIS_METRIC_TLV_WIDE            \
     |ISIS_METRIC_SPF_NARROW|ISIS_METRIC_SPF_WIDE)

#define IS_ISIS_METRIC(L,M)                                 \
    ((L)->metric_style == ISIS_METRIC_ ## M)
#define ISIS_METRIC_TLV_CHECK(L,M)                         \
    ((L)->metric_style & ISIS_METRIC_TLV_ ## M)
#define ISIS_METRIC_SPF_CHECK(L,M)                         \
    ((L)->metric_style & ISIS_METRIC_SPF_ ## M)
#define ISIS_METRIC_SET(L,M)                               \
    ((L)->metric_style = ISIS_METRIC_ ## M)

  /* Topology type. */
  u_char topology_type;
#define ISIS_TOPOLOGY_TLV_SINGLE                 (1 << 0)
#define ISIS_TOPOLOGY_SPF_SINGLE                 (1 << 1)
#define ISIS_TOPOLOGY_TLV_MULTI                  (1 << 2)
#define ISIS_TOPOLOGY_SPF_MULTI                  (1 << 3)
#define ISIS_TOPOLOGY_TLV_PROTOCOL               (1 << 4)
#define ISIS_TOPOLOGY_SPF_PROTOCOL               (1 << 5)

#define ISIS_TOPOLOGY_SINGLE                              \
    (ISIS_TOPOLOGY_TLV_SINGLE|ISIS_TOPOLOGY_SPF_SINGLE)
#define ISIS_TOPOLOGY_MULTI                              \
    (ISIS_TOPOLOGY_TLV_MULTI|ISIS_TOPOLOGY_SPF_MULTI)
#define ISIS_TOPOLOGY_MULTI_TRANSITION                    \
    (ISIS_TOPOLOGY_TLV_SINGLE|ISIS_TOPOLOGY_SPF_SINGLE      \
     |ISIS_TOPOLOGY_TLV_MULTI|ISIS_TOPOLOGY_SPF_MULTI)
#define ISIS_TOPOLOGY_PROTOCOL                              \
    (ISIS_TOPOLOGY_TLV_PROTOCOL|ISIS_TOPOLOGY_SPF_PROTOCOL)
```



```
#define IS_ISIS_TOPOLOGY(L, T)                                      \
    ((L)->topology_type == ISIS_TOPOLOGY_ ## T)
#define ISIS_TOPOLOGY_TLV_CHECK(L, T)                                \
    ((L)->topology_type & ISIS_TOPOLOGY_TLV_ ## T)
#define ISIS_TOPOLOGY_SPF_CHECK(L, T)                               \
    ((L)->topology_type & ISIS_TOPOLOGY_SPF_ ## T)
#define ISIS_TOPOLOGY_ADJ_CHECK(L, T)                                \
    ISIS_TOPOLOGY_SPF_CHECK (L, T)
#define ISIS_TOPOLOGY_SET(L, T)                                     \
    ((L)->topology_type = ISIS_TOPOLOGY_ ## T)

  /* LSPDB. */
  struct ls_table *lspdb;

  /* IPv4 protocol data. */
  struct isis_level_proto proto[ISIS_PROTO_INDEX_MAX];

  /* Vector of IS-neighbor map. */
  vector is_map_vec;

 /* Timestamps. */
  struct pal_timeval tv_spf;            /* SPF calculation last performed. */
  struct pal_timeval tv_spf_curr;       /* Current SPF calculation. */

  /* PRC Timestamps. */
  struct pal_timeval tv_prc;            /* PRC calculation last performed. */
  struct pal_timeval tv_prc_curr;       /* Current PRC calculation. */

  /* TLV and MAP. */
  struct isis_tlv *tlv_auth_info;       /* Authentication Info TLV. */
#ifdef HAVE_MD5
  struct isis_tlv *tlv_hmac_md5_auth_info; /* MD5 Authentication Info TLV.*/
#endif /* HAVE_MD5*/

  u_char prc_flags;
#define ISIS_PRC_CALC                (1 << 0)

  /* Threads. */
  struct thread *t_spf_calc;            /* SPF calculation timer. */
  struct thread *t_reach_map;           /* Prefix Map timer. */
  struct thread *t_max_seqnum;          /* Wait timer for
ExceedMaxSeqNumber.*/
  struct thread *t_prc_calc;            /* PRC calculation timer. */
#ifdef HAVE_RESTART
  struct thread *t_restart;             /* Restart timer. */
#endif /* HAVE_RESTART */

  /* Statistics. */
  u_int32_t auth_type_fails;            /* isisSysStatAuthTypeFails. */
  u_int32_t auth_fails;                 /* isisSysStatAuthFails. */
  u_int32_t corrupted_lsps;             /* isisSysStatCorrLSPs. */
  u_int32_t lspdb_overloaded;           /* isisSysStatLSPDbaseOloads. */
  u_int32_t man_addr_drop_area;         /* isisSysStatManAddrDropFromAreas. */
  u_int32_t exceed_max_seqnums;         /* isisSysStatAttmptToExMaxSeqNums. */
  u_int32_t seqnum_skips;               /* isisSysStatSeqNumSkips. */
  u_int32_t lsp_purges;                 /* isisSysStatOwnLSPPurges. */
  u_int32_t id_len_mismatches;          /* isisSysStatIDFieldLenMismatches. */
```

```
  u_int32_t partition_changes;          /* isisSysStatPartChanges. */
  u_int32_t spf_calc_count;             /* isisSysStatSPFRuns. */
  u_int32_t prc_calc_count;             /* isisSysStatPRCRuns. */
};
```

isis_lsp_header

This data structure contains IS-IS LSP header information.

Definition

```
struct isis_lsp_header
{
  /* PDU Length. */
  u_int16_t pdu_length;

  /* Remaining Lifetime. */
  u_int16_t lifetime;

  /* LSP ID. */
  union
  {
    u_char lsp_id[ISIS_LSPID_LENGTH];
    struct isis_lspid lspid;
  } ul;

  /* Sequence Number. */
  u_int32_t seqnum;

  /* Checksum. */
  u_int16_t cksum;

  /* P, ATT, OL and IS-Type bits. */
 u_char flags;
#define ISIS_LSP_BIT_ISTYPE_L1           (1 << 0)
#define ISIS_LSP_BIT_ISTYPE_L2           (1 << 1)
#define ISIS_LSP_BIT_OVERLOAD            (1 << 2)
#define ISIS_LSP_BIT_ATTACHED_DEFAULT    (1 << 3)
#define ISIS_LSP_BIT_ATTACHED_DELAY      (1 << 4)
#define ISIS_LSP_BIT_ATTACHED_EXPENSE    (1 << 5)
#define ISIS_LSP_BIT_ATTACHED_ERROR      (1 << 6)
#define ISIS_LSP_BIT_PARTITION_REPAIR    (1 << 7)
};
```

isis_area_addr

This data structure contains IS-IS manual area address information.

Definition

```
struct isis_area_addr
{
  u_char length;
  u_char address[ISIS_AREA_ADDR_LEN_MAX];
};
```

```
/* Manual Area Addresses. */
struct isis_recv_area_addr
{
  u_char length;
  u_char address[ISIS_AREA_ADDR_LEN_MAX];
  u_int32_t count; /* Count of LSP's using this structure */
};
```

isis_dis_id

This data structure contains both source and circuit identifiers.

Definition

```
struct isis_dis_id
{
  /* Source ID. */
  u_char source_id[ISIS_SYSID_LENGTH];
  /* Circuit ID. */
  u_char circuit_id;
};
```

isis_tlv

This data structure contains time length values.

Definition

```
struct isis_tlv
{
  /* TLV type. */
  u_char type;
  /* TLV length. */
  u_char len;
  /* TLV length not committed. */
  u_char len_tmp;
  /* TLV flags. */
  u_char flags:3;
#define ISIS_TLV_SELF           (1 << 0)
#define ISIS_TLV_LSP_ATTACHED   (1 << 1)
  /* Allocated slot -- (slot + 1) * 8 octets. */
  u_char slot:5;
#define ISIS_TLV_DATA_BYTES(T)          (((T)->slot + 1) * 8)
#define ISIS_TLV_DATA_SLOTS(O)          (((O) - 1) / 8)
  /* Pointer to parent LSP. */
  struct isis_lsp *lsp;
  /* TLV body. */
  u_char *data;
};
```

isis_header

This data structure contains IS-IS PDU header information.

```
struct isis_header
{
```

```
    u_char pid;                  /* Initial Protocol Identifier. */
    u_char length;               /* Header length. */
    u_char version_proto_id;     /* Version/Protocol ID Extension. */
    u_char id_length;            /* ID Length -- 0 stands for 6. */
    u_char pdu_type;             /* PDU type. */
    u_char version;              /* Version. */
    u_char reseved;              /* Reserved. */
    u_char max_area_addresses;   /* Maximum Area Addresses -- 0 stands for 3. */
  };
```

Databases

The following subsection describes the databases associated with IS-IS.

Interface Table

Interface information is stored in the IS-IS interface table, which belongs to the IS-IS instance. All IS-IS enabled interfaces are stored in this table.

Neighbor Database

For broadcast interfaces, IS-IS adjacency (IS-IS neighbor) is kept in a table that belongs to the IS-IS logical interface struct. 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 IS-IS instance structure, which stores neighbor information belonging to all interfaces.

LSP Database

The LSP database is the core of IS-IS routing. All link-state information advertised by neighbors in the same domain or area is stored in this database. LSP databases for level-1 and level-2 structures are separately maintained and belong to the IS-IS instance level structure.

IPv4 Routing Table

Whenever the LSP database is updated, each level triggers to perform SPF calculation. As a result, it generates IP routing information and stores it in the IPv4 routing table. After building the routing table, IS-IS sends IPv4 routing information to the NSM to install the routes into FIB.

Hostname Table

The dynamic hostname is delivered with LSP flooding. This information is stored into a dynamic hostname table shared by all IS-IS instances. The table is maintained to show the canonical name, instead of the system ID.

CHAPTER 3 Restart Signaling

This chapter discusses the ZebOS-XP implementation of the IS-IS restart signaling feature.

Overview

The Intermediate System to Intermediate System (IS-IS) routing protocol is a link state intra-domain routing protocol. Previously, a temporary disruption of routing occurred whenever an IS-IS router restarted due to events in both the restarting router and the neighbors of the router. Neighbors of the restarting router detected the restart event and cycle their adjacencies with the restarting router through the down state. The cycling of the adjacency state caused the neighbors to regenerate the LSPs (Link State Protocols) that described the adjacency concerned. This caused a temporary disruption of routes passing through the restarting router.

RFC 5306 helps avoid or minimize the disruption of routes. RFC 5306 describes a mechanism that instructs a restarting router to signal its neighbors that it is restarting, thus allowing them to reestablish their adjacencies without cycling through the down state, while still correctly initiating database synchronization. In addition, it describes a mechanism for a restarting router to determine when it has achieved LSP database synchronization with its neighbors, and optimizes LSP database synchronization while minimizing transient routing disruption when a router starts.

System Architecture

Typically, an IS-IS router automatically routes around a restarting router. With restart signaling, a restarting router announces the grace period to its neighboring routers by storing the time in the "holding time" field of the IS-IS hello packet. Neighboring routers continue to announce the restarting router as if it were still adjacent. When the restarting router comes back online, neighboring routers are notified to not change the state of the adjacency when a IS-IS hello packet with restart TLV is received, so that no SPF recalculation occurs on neighboring routers.

A "starting router" refers to a router where its control function has either commenced operations for the first time or has resumed operations, but the forwarding functions have not been maintained in a prior state. An SA bit is used by the starting router to request that its neighbor suppress advertisement of the adjacency to the starting router in the neighbor LSPs. When a router receives an IIH (IS-IS Hello PDUs) with the restart TLV having the SA bit set, if there is an adjacency on this interface as "UP" with the same system ID and with the same source LAN address, then the router must suppress advertisement of the adjacency to the neighbor in its own LSPs. The neighbor advertisement must continue to be suppressed until an IIH with the SA bit clear is received.

Note: Restart signaling is possible when the network topology is stable, and the restarting router retains its forwarding tables.

Restart Mode

A router that sends out an IS-IS hello packet containing a grace-period value in the hold-time field begins the restart process. The interval from when the IS-IS hello packet is sent until the synchronization of the restarting router’s LSP database with neighboring routers is called the restart signaling mode. When in restart signaling mode, the router:

- does not originate LSPs. Neighboring routers calculate routes using the LSPs sent prior to entering restart signaling mode.
- does not install IS-IS routes into the forwarding tables, since it relies on the forwarding entries extant prior to entering restart signaling mode.
- retains its LAN-ID for broadcast interfaces.

Entering Restart Mode

A router enters restart mode when a network administrator gives the `restart isis graceful` command (see the *Intermediate System to Intermediate System Command Reference)*. The restarting router uses the `nsm_client_send_preserve_with_val` function to preserve the forwarding table. The router also preserves (in non-volatile memory) the LAN-ID for broadcast interfaces. When the forwarding table and LAN-ID are preserved, the restarting router sends the IS-IS hello packet with grace period, one for each IS-IS interface. The restart signaling mode then proceeds with reloading and restarting the router.

Exiting Restart Mode

The restart signaling mode is exited when the LAN-ID changes on one broadcast interface during restart signaling and when the synchronization of the restarting router's LSP database is not finished before the synchronization timer (T2) expires. (Refer to Timers for more information).

Helper Mode

A helper router is a router that does not change the adjacency state with a restarting router, but instead sends CSNPs and LSPs. Entrance to the helper mode is contingent upon several conditions, not the least of which is local policy. The following criteria must be met:

- Helper router has full adjacency with the restarting router
- There are no changes to the LSP database since the initiation of the restart
- The grace period (given in the IS-IS hello packet) has not expired
- Local policies are followed

Note: Any router can act as a helper router for multiple restarting routers. Grace periods can be updated if a subsequent IS-IS Hello packet is received from the restarting router.

A helper router remains in helper mode until a successful restart of the router or when the grace period expires. A successful restart of the router is when an IS-IS hello packet (without the restart request bit set in the restart TLV) is received from the restarting router.

Timers

ZebOS-XP maintains three different timers to implement the restart signaling:

T1. This timer is maintained for each interface, and indicates the time after which ZebOS-XP repeats an unacknowledged restart attempt. The T1 timer is configured with the `isis restart-hello-interval` command. The default value is 3 seconds.

T2. This timer is maintained for each LSP database present in the each level. This is the maximum time that the system will wait for LSP database synchronization. The T2 timer is configured with the `isis restart-timer` command. The default value is 60 seconds.

T3. This timer is maintained for the entire system. It indicates the time after which the router will declare that it has failed to achieve database synchronization. The T3 timer is configured with the `isis restart grace-period` command. The default value is 65535 seconds.

Note: See the *Intermediate System to Intermediate System Command Reference* for more information on all of these commands.

Grace Period

When a router exits, it resumes normal routing duties depending on the restarting result. The function `isis_restart_success_all` accomplishes the following actions when restarting succeeds:

- changes the interface state to DIS or nonDIS
- re-originates its regular LSPs
- re-originates its pseudo node LSPs if it is the DIS
- recalculates the routes, installing results into the system forwarding table
- removes remnant entries from the system forwarding table

The function `isis_restart_fail_all ()` accomplishes these actions when restarting fails:

- changes the interface state to Down, and Up, once again
- does ordinary starting

The function isis_restart_fail_all_overload accomplishes these actions when the T3 timer expires before the T2 timers: It sets the overload bit on its self-LSPs and floods them all.

Restart Signaling Source Code

The code for ZebOS-XP IS-IS restart signaling can be found in the `isis_restart.c` and `isis_restart.h` files, and in the `isis_nsm.c` file. The `isis_nsm.c` file adds restart support to the NSM.

isis_restart.h. This file contains the data definitions and structures for the restart signaling. .

isis_restart.c. This file contains the procedural code for restart signaling. .

isis_cli.c. This file contains the CLI statements for the CLI commands. See the *Intermediate System to Intermediate System Command Reference* for details. .

isis_nsm.c. This file contains the support code for restart signaling bracketed by `#ifdef HAVE_RESTART... #endif /* HAVE_RESTART */` .

API

The following subsection describes the API for the ISIS restart signaling feature.

nsm_client_send_preserve_with_val

Syntax

```
s_int32_t
nsm_client_send_preserve_with_val (struct nsm_client *nc, int restart_time,
                                   u_char *restart_val,
                                   u_int16_t restart_length)
```

Input Parameters

`*nc`	NSM client.

`restart_time`	Restart time.
`restart_length`	Restart length.

Output Parameters

`*restart_val`	Restart value.

Return Value

isis_restart_fail_all_overload

Syntax

```
void
isis_restart_fail_all_overload (struct isis_master *im)
```

Input Parameters

`*im`	Pointer to the IMI master.

Output Parameters

None

Return Value

0

1

isis_restart_level_triggered_check

Syntax

```
void
isis_restart_level_triggered_check (struct isis_level *il)
```

Input Parameters

`*il`	Pointer to level context of IS-IS instance.

Output Parameters

None

Return Value

0

isis_restart_level_timer

Use this function restarts the T2 level timer.

Syntax

```
int
isis_restart_level_timer (struct thread *thread)
```

Input Parameters

`*thread`	Pointer to the thread structure.

Output Parameters

None

Return Value

0

isis_restart_suppress_adjacency_set

Syntax

```
int
isis_restart_suppress_adjacency_set (u_int32_t vr_id)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

isis_is_reach_map_tlv_psn_set

Suppression of adjacency in LSPs is done in function

Syntax

```
void
isis_is_reach_map_tlv_psn_set (struct isis_tlv *tlv,
                               struct isis_is_reach_map *map)
```

Input Parameters

`*tlv`	Pointer to the time length value.
`*map`	Pointer to the reach map.

Output Parameters

None

Return Value

tlv

isis_is_reach_map_tlv_reg_set

Syntax

```
void
isis_is_reach_map_tlv_reg_set (struct isis_tlv *tlv,
                               struct isis_is_reach_map *map)
```

Input Parameters

`*tlv`	Pointer to the time length value.
`*map`	Pointer to the reach map.

Output Parameters

None

Return Value

0

tlv

isis_restart_tlv_update

Use this function retrieve the neighbor system ID of the restarting router.

Syntax

```
struct isis_tlv *
isis_restart_tlv_update (struct isis_if_level *ifl)
```

Input Parameters

`*ifl`	Pointer to the parent interface level.

Output Parameters

None

Return Value

0

tlv

isis_restart_if_level_init

Syntax

```
int
isis_restart_if_level_init (struct isis_if_level *ifl)
```

Input Parameters

`*ifl`	Pointer to the parent interface level.

Output Parameters

None

Return Value

0

1

isis_restart_level_set

Syntax

```
void
isis_restart_level_set (struct isis_level *il)
```

Input Parameters

`*il`	Pointer to level context of IS-IS instance.

Output Parameters

None

Return Value

0

isis_nsm_recv_service

This function is for when a service reply is received.

Syntax

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

Input Parameters

`*il`	Pointer to level context of IS-IS instance.

Output Parameters

None

Return Value

0

-1

isis_ism_timer_set

Syntax

```
void
isis_ism_timer_set (struct isis_if_level *ifl, int event)
```

Input Parameters

`*ifl`	Pointer to the parent interface level.
`event`	Pointer to the event callback.

Output Parameters

None

Return Value

isis_tlv_put_hello_3way_adj

This function instructs the adjacency 3-way state to initiate when restarting the router for P2P network type

Syntax

```
void
isis_tlv_put_hello_3way_adj (struct stream *s, struct isis_if_level *ifl)
```

Input Parameters

`*s`	Pointer to the IMI server
`*ifl`	Pointer to the parent interface level.

Output Parameters

None

Return Value

1

0

isis_restart_lan_tlv_check

Syntax

```
int
isis_restart_lan_tlv_check (struct isis_nbr_level *nl, struct isis_packet *pkt)
```

Input Parameters

`*nl`	Pointer to the IS-IS level for REG-LSP.
`*pkt`	Pointer to packet buffer.

Output Parameters

None

Return Value

1

0

isis_restart_timer_update

Syntax

```
void
isis_restart_timer_update (struct isis_master *im, u_int16_t holding_time)
```

Input Parameters

`*im`	Pointer to the IMI master.
`holding_time`	Holding time.

Output Parameters

None

Return Value

1

0

isis_lan_nsm_neighbor_up

This function is a neighbor up handler.

Syntax

```
void
isis_lan_nsm_neighbor_up (struct isis_nbr_level *nl)
```

Input Parameters

`*nl`	Pointer to the IS-IS level for REG-LSP.

Output Parameters

None

Return Value

1

0

isis_lan_nsm_neighbor_up

This function is a neighbor up handler.

Syntax

```
void
isis_lan_nsm_change_state (struct isis_nbr_level *nl, int state)
```

Input Parameters

`*nl`	Pointer to the IS-IS level for REG-LSP.
`state`	ISM state

Output Parameters

None

Return Value

1

0

isis_lsp_flags_get

Syntax

```
u_char
isis_lsp_flags_get (struct isis *top, int index,
                    u_char psn_id, u_char lsp_num)
```

Input Parameters

`*top`	ISIS instance lookup.
`index`	ISIS index.
`psn_id`	Represents the pseudonode ID.
`lsp_num`	LSP number

Output Parameters

None

Return Value

1

0

CHAPTER 4 IS-IS Traffic Engineering

This chapter discusses Traffic Engineering (TE) support in the IS-IS protocol module.

Overview

ZebOS-XP IS-IS supports intra-area traffic engineering, where TE information is flooded in the respective level. An IS-IS router can be configured as Level-1, Level-2 or Level-1-2 router. If a router is Level-1 or Level-2, it maintains LSDB (link state database) only for that level. Therefore, for Level-1 or Level-2 Routers, TE information can be flooded in their respective levels and TE tunnels can be set up in their respective levels only.

However, a Level-1-2 router maintains two LSDBs, one for each level and therefore can compute route for destination in its area in Level-1 or to a Level-2 router. ZebOS-XP allows TE information to be flooded in both Level-1 and Level-2. This allows Level-1-2 router to set up tunnels in Level-1 as well as in Level-2. ZebOS-XP does not support inter-area TE for IS-IS. The following is a sample topology of IS-IS TE support.

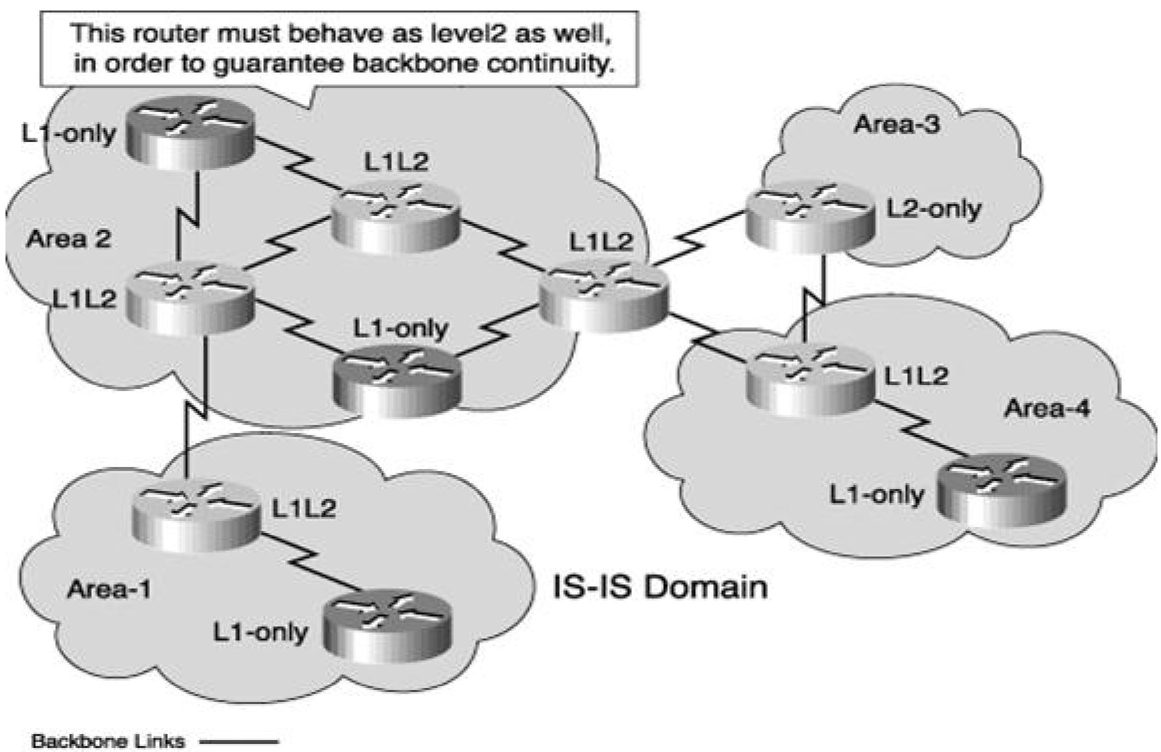

Figure 4-1: Sample Topology

System Architecture

IS-IS Traffic Engineering feature requires MPLS traffic engineering to be enabled. For Level-1 or Level-2 Router, MPLS traffic engineering should be configured only its respective level. For IS-IS Level-1-2 routers, MPLS traffic engineering can be configured on both Level-1 and Level-2 routers.

CSPF (Constrained Shortest Path First) server supports calculation of routes for both Level-1 and Level-2 destination depending on whether IS-IS router is Level-1 or Level-2 Router.

For Level-1-2 Router, CSPF server supports to calculate paths to egress in Level-1 as well as in Level-2. For example, if a tunnel is set up to a Level 1 router, CSPF computes a route based on the TE database in Level 1. Conversely, if another tunnel is set up to a Level 2 router, CSPF will compute route based on TE database in Level 2.

ZebOS-XP limits one CSPF server running in multiple IS-IS instances. CSPF supports only one instance of IS-IS. In addition, only IPV4 is supported for CSPF calculation.

Note: Before configuring TE requirements for both IS-IS TE and CSPF extension on Level-1 and Level-2 routers, the user must enable the CLI command "mpls traffic-eng" on both Level-1 and Level-2. Refer to the Intermediate System to Intermediate System Command Line Interface Reference Guide for more information on this command.

Supported APIs

The following subsection describes the supported APIs for the TE feature.

isis_mpls_traffic_eng_set

Use this function to enable traffic engineering in both level-1 and level-2 routers.

Syntax

```
int
isis_mpls_traffic_eng_set (u_int32_t vr_id, char *tag, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET` if the Wide metric is not set.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_TE_NOT_ENABLED` if given level is not configured as TE-enabled.

isis_mpls_traffic_eng_unset

Use this function to disable traffic engineering for both level-1 and level-2 routers.

Syntax

```
int
isis_mpls_traffic_eng_unset (u_int32_t vr_id, char *tag, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_TE_NOT_ENABLED` if given level is not configured as TE-enabled.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_cspf_set

This function creates an IS-IS CSPF server.

Syntax

```
int
isis_cspf_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_TE_NOT_ENABLED` if given level is not configured as TE-enabled.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_CSPF_INSTANCE_EXIST`

`ISIS_API_SET_ERR_CSPF_ENABLE_FAILED` if the call failed for other reasons.

`ISIS_API_SET_ERR_CSPF_DISABLE_FAILED` if the call failed for other reasons.

`ISIS_CSPF_SET AND ISIS_CSPF_UNSET`

isis_cspf_finish

This function cleans up a CSPF server.

Syntax

```
int
isis_cspf_finish (struct isis *top)
```

Input Parameters

`top`	ISIS instance lookup.

Output Parameters

None

Return Value

0

isis_level_finish

This function removes the isis_cspf_finish calling, since the CSPF server is moved to the ISIS instance structure.

Syntax

```
void
isis_level_finish (struct isis *top, int index)
```

nput Parameters

`top`	ISIS instance lookup.
`index`	ISIS index.

Output Parameters

None

Return Value

None

isis_instance_finish

This function remove of calling isis_cspf_finish, since CSPF server is moved to the ISIS instance.

Syntax

```
int
isis_instance_finish (struct isis *top)
```

nput Parameters

top	ISIS instance lookup.

Output Parameters

None

Return Value

0

isis_cspf_init

This function pass an ISIS instance as CSPF/parent when creating or finishing a CSPF server.

Syntax

```
int
isis_cspf_init (struct isis *top)
```

nput Parameters

top	ISIS instance lookup.

Output Parameters

None

Return Value

0

-1

isis_cspf_lsp_compute

This function finds an ISIS route in an ISIS instance routing table. Use the selected ISIS route path to get an ISIS level. In addition, use the ISIS-level TE information to calculate a CSPF LSP.

Syntax

```
int
isis_cspf_lsp_compute (struct cspf *cspf, struct cspf_lsp *lsp)
```

nput Parameters

cspf	CSPF
lsp	Link-state packet

Output Parameters

None

Return Value

CSPF_FAILURE

CSPF_SUCCESS

CHAPTER 5 Overload Bit

ISIS is designed to move information efficiently within a network. It accomplishes this by determining the best route for datagrams through a packet-switched network. In ZebOS-XP, if a router is not able to add more than 256 leaked LSP's in its database, then routes are dropped. In addition, the router does not notify the other routers that it is overloaded.

Overview

When a router runs out of system resources (memory or CPU), it cannot store the link-state PDU into the database or run shortest path first (SPF). In this situation, the router must alert the other routers within its area by setting a particular bit in its link-state packets (LSPs). When other routers detect that this bit is set, they do not use this router to transit traffic. However, they use it for packets destined to the overloaded router's directly connected networks and IP prefixes.

In ISIS, a router immediately floods its own LSP even before sending complete sequence number PDU (CSNP) packets. The overload bit is thereby used to advise the rest of the network not to route transit traffic through the overloaded router.

For each LSP, the ISO/IEC 10589:1992 defines a special bit called LSP Database Overload Bit. If there is a network mis-configuration or a transitory condition, it is possible that there may be insufficient memory resources available to store received link state PDUs. When this takes place, an Intermediate System (IS) needs to take steps to ensure that if its LSP database becomes inconsistent with other IS's, then these IS's do not rely on forwarding paths through the overloaded IS. When an IS is in an overloaded condition, it sets the overload bit in the non-pseudo node LSP fragment 0 that it generates.

In addition, even though the other IS's do not use the overloaded IS as a transit router, they are still able to reach the directly attached end systems. During this time, directly connected interfaces (as well as IP prefixes) are reachable.

With the ZebOS-XP ISIS implementation, the overload bit feature is applicable when a level 1 route leaks into level 1 or when a level 2 route leaks into level 1. In this process, if the L2/L1 LSP's exceed the ISIS_LSP_NUM_MAX 255 value, the overload bit is set into the first L2/L1 LSP (that is, LSP Zero) and then advertised to the neighbor routers.

System Configuration

The following subsection describes how to set and unset the overload bit.

Setting the Overload Bit

A level 1 router may leak its database into level 2 or a level 2 router can leak its database into level 1. At times, the database leak can cause the router to generate more than 256 LSPs in a level. In such a case, if the routes cannot be added into the LSP, the router may become overloaded.

When an LSP cannot be stored, the LSP is ignored and a Waiting State is started. In addition, a timer is started for waitingTime (default is 60 seconds) and the IS generates and floods its own LSP with a "zero" LSP number with the LSP database overload bit set. This prevents the IS from being regarded as a forwarding path by other IS's. It is possible that, although there are sufficient resources to store an LSP and permit the operation of the update process on that LSP, the decision process may subsequently require further resources in order to complete. If these resources are unavailable, the IS enters a waiting state until resources become available and the waitingTime has elapsed since the last LSP was ignored by the update process.

Actions in Level 1 Waiting State

The following takes place while in Level 1 “waiting” state:

- If a Link State PDU cannot be stored, the IS ignores it and restarts the waitingTime timer.
- IS continues to run the decision and forwarding processes as normal.
- When the waitingTime timer expires, the IS does the following:
  - Generates an ISPL1DatabaseOverload (recovered) event.
  - Clears the LSP database overload bit in its own level 1 LSP with zero LSP number and re-issues it.
  - Sets the l1State to “On”.
  - Resumes normal operation.

Actions in Level 2 Waiting State

The following takes place while in Level 2 “waiting” state:

- If a link state PDU cannot be stored, the IS ignores it and restarts the waitingTime timer.
- IS continues to run the decision and forwarding processes as normal.
- When the waitingTime timer expires, the IS does the following:
  - Generates an ISPL2DatabaseOverload (recovered) event.
  - Clears the LSP database overload bit in its own Level 2 LSP with zero LSP number and re-issues it.
  - Sets the l2State to “On”.
  - Resumes normal operation.

Neighbor Functionality in the Overloaded Router

When a neighbor receives an LSP with overload bit set from the overloaded router, it installs it into its database. However, it does not install the LSP routes into its FIB so that transit traffic is not forwarded to the overloaded router. When the waitingTime timer of the overloaded router expires, the router sends the LSP zero with the overload bit cleared to its neighbors. SPF calculation is done after the neighbor receives this LSP. Since the overload bit is cleared, the routes are installed and the transit traffic is sent to the router.

Overload Bit with BGP Converges

ISIS overload bit can also be configured to work with BGP convergence. When BGP converges, ISIS overload bit allows a router to automatically disable the overload bit. This is useful to Internet service providers who run both BGP and ISIS to avoid black-hole scenarios, which decreases data loss associated with the deterministic black-holing of packets during transient network conditions. This is better for stability and availability for routers to build their BGP routing tables when they are not fully participating in packet forwarding.

ISIS and BGP routing are mutually dependent upon each other: if both do not converge simultaneously, traffic is black-holed. BGP runs on top of TCP, and in order for TCP to function, valid internal routes are required. IGP provides this information; in this case, the IGP ISIS.

BGP Converges Overview

The following network diagram depicts a high-level overview of the system and black-hole scenario the overload bit is designed to address.

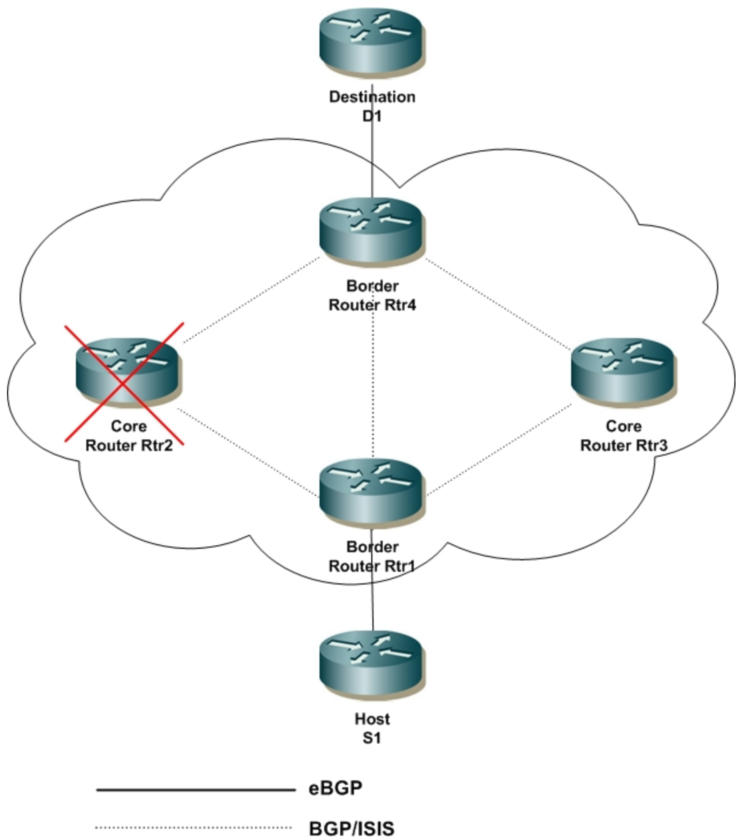

Figure 5-1: BGP/ISIS Network

In the preceding graphic, host S1 transmits data to destination D1 through a primary path of Rtr1, Rtr2, and Rtr4. Routers 1, 2, 3 learn reachability to destination D1 through BGP from Rtr4. If the core router Rtr2 goes down, other routers within the routing domain will select an alternative path to reach their destinations. The alternate path would be Rtr1, Rtr3, and Rtr 4. ISIS then synchronizes its link-state database.

When the previously-failed router (core router Rtr2) becomes available again, it has seconds before the path that had previously transited the router are again selected as the optimal path by the ISIS router. As a result, forwarding tables are updated and packets are again forwarded along the path. The external destination reachability information (for example, learned via BGP) is not yet available to the router and packets bound for destinations not learned through the IGP are discarded. Core router Rtr2 does not have the transit BGP routes to know where to forward the traffic, since the BGP sessions are not yet established. This is the black-hole state in which packets flowing into Core router Rtr2 have no place to go. Rtr2 discards the packets received from Rtr1 destined to D1.

From Figure 5-1 on page 53, the Rtr2 LSP has the overload bit set, when Rtr1 computes the SPF, it does not use Rtr2 as the transient node, since the shortest path to Rtr-4 is Rtr-1, Rtr3, and Rtr4. Overload bit is able to configure the router to automatically disable the overload bit when BGP converges.

BGP Connections

Once a BGP connection is established using open messages, BGP peers initially use update messages to send each other the routing information. It then enters a routine in which the BGP session is maintained. However, update messages are sent only when required. To ensure that connection does not terminate when there are no update messages for awhile, each peer periodically sends a BGP keepalive message. BGP has converged when keepalives are received from all BGP neighbors.

If the peers are not in the established state in the BGP peer FSM state, a check is made to determine if a peer has changed from the idle to established state. Then, a check is made to determine if a keepalive message was exchanged with that peer, which increments the `neighbors_converged` counter. Once this condition is satisfied for all peers, all neighbors are converged.

ZebOS-XP BGP Architecture

ZebOS-XP supports configuring a router on startup to advertise its LSP with the overload bit for a specific amount of time after a reload. When the configured timer interval expires, the overload bit clears and the LSP is re-flooded. In addition, overload bit allows the router to be configured with a `wait-for-bgp` parameter, that ensures the router does not receive transit traffic while the routing protocol is converging. With this feature, the router can be configured either by setting the overload bit for a fixed amount of time after reload or by configuring the router to not receive transit traffic while the BGP routing protocol is still converging.

The trigger for setting the overload bit after ISIS reload is the configuration of the command, `set-overload-bit on-startup wait-for-bgp`.When `wait-for-bgp` is configured, an overload bit is set in the ISIS LSP immediately after reload and starts the overload expiration timer upon receiving a signal from NSM that BGP has converged to clear the overload bit. If the BGP process has already converged or if the BGP process is not running, the overload bit is cleared in the LSP, and the overload expiration timer is cleared.

See the *Border Gateway Protocol Command Reference* for more about the `set-overload-bit` command.

Overload Bit APIs

The following subsection describes the APIs that support the overload bit feature.

isis_parse_overload_option

This call sets the parse the options for the set-overload-bit command.

Syntax

```
int
isis_parse_overload_option (bool_t args, bool_t startup_val,
                             bool_t wait_for_bgp, bool_t suppress_external,
                             bool_t suppress_interlevel, u_char *option)
```

Input Parameters

`args`	TRUE indicates there are arguments passed with set-overload-bit. FALSE indicates no arguments passed with set-overload-bit.
`startup_val`	
`wait_for_bgp`	
`suppress_external`	
`suppress_interlevel`	
`*option`	

Output Parameters

None

Return Values

1

0

isis_dynamic_overload_set

This call sets the overload bit state and updates the LSPs.

Syntax

```
void
isis_dynamic_overload_set (struct isis_level *il)
```

Input Parameters

`*il`	Represents the isis level (that is, level-1 or level-2) based structure.

Output Parameters

None

Return Values

None

isis_dynamic_overload_unset

This call unsets the overload bit state and updates the LSPs.

Syntax

```
void
isis_dynamic_overload_unset (struct isis_level *il)
```

Input Parameters

`*il`	Represents the isis level (that is, level-1 or level-2) based structure.

Output Parameters

None

Return Values

None

isis_lsp_tlv_get_gap

If a router is overloaded (that is, the LSP exceeds the value for the ISIS_LSP_NUM_MAX parameter of 255), then the overload bit is set into the LSP as 0 and that LSP is sent to its neighbors and a waiting timer is started for waitingTime (default 60 seconds). The waiting time is configurable in the range of 1 to 65535.

If an LSP is received before the waiting time expires, then the LSP is ignored. Once the timer expires the overload bit is cleared and the LSP 0 is triggered to its neighbors.

Syntax

```
struct isis_tlv *
isis_lsp_tlv_get_gap (struct isis_level *il, u_char psn_id,
                      u_char type, u_char len)
```

Input Parameters

`*il`	Represents the isis level (that is, level-1 or level-2) based structure.
`psn_id`	Represents the pseudonode ID.
`type`	Represents the TLV type.
`len`	Represents the length of the TLV.

Output Parameters

None

Return Values

Null

lsp

isis_spf_ipv4_reach_process

This API checks for an LSP with an overload bit, as well as the LSP with the same system ID as the LSP with the overload bit. These LSPs with routes should not be installed in the FIB, but can be in the LSP database.

Syntax

```
void
isis_spf_ipv4_reach_process (struct isis_level_proto *ilp,
                             struct isis_vertex *v, u_char type)
```

Input Parameters

`*ilp`	Represents level based ipv4/ipv6 protocol data.
`*v`	Represents isis vertex.
`type`	Represents the TLV type.

Output Parameters

None

Return Values

1

0

isis_spf_ipv6_reach_process

This API checks for an LSP with an overload bit, as well as the LSP with the same system ID as the LSP with the overload bit. These LSPs with routes should not be installed in the FIB, but can be in the LSP database.

Syntax

```
void
isis_spf_ipv6_reach_process (struct isis_level_proto *ilp,
                             struct isis_vertex *v, u_char type)
```

Input Parameters

`*ilp`	Represents level based ipv4/ipv6 protocol data.
`*v`	Represents isis vertex.
`type`	Represents the TLV type.

Output Parameters

None

Return Values

1

0

isis_overload_timer

This call sets the overload bit timer.

Syntax

```
int
isis_overload_timer (struct thread *thread)
```

Input Parameters

`*thread`	Thread value.

Output Parameters

None

Return Values

0

isis_nsm_wait_for_bgp_set

This call communicates from the NSM client in ISIS to the NSM server to send the overload bit `wait_for_bgp_set` request message to NSM. The NSM client is a link to NSM. Each protocol module, such as ISIS and BGP, has an instance of the NSM client. For each message received from NSM, the NSM client installs parsers for the messages. The protocols register callback functions that will be called by the respective parse

Syntax

```
void
isis_nsm_wait_for_bgp_set (struct isis_master *im, u_int16_t flag)
```

Input Parameters

`*im`	Represents the ISIS master structure.
`flag`	Boolean flag.

Output Parameters

None

Return Values

None

isis_nsm_wait_for_bgp_set

This call sends a wait for BGP set request to NSM.

Syntax

```
void
isis_nsm_wait_for_bgp_set (struct isis_master *im, u_int16_t flag)
```

Input Parameters

`*im`	Represents the ISIS master structure.
`flag`	Boolean flag.

Output Parameters

None

Return Values

None

isis_overload_bit_wait_for_bgp_unset

This call sends a wait for BGP unset a request to NSM.

Syntax

```
void
isis_overload_bit_wait_for_bgp_unset (struct isis_master *im)
```

Input Parameters

`*im`	Represents the ISIS master structure.

Output Parameters

None

Return Values

None

isis_nsm_redistribute_set

This call sets a redistribute request to NSM.

Syntax

```
void
isis_nsm_redistribute_set (struct isis_master *im, u_char proto, int type)
```

Input Parameters

`*im`	Represents the ISIS master structure.
`proto`	Defines IPv4/IPv6.
`type`	Message type.

Output Parameters

None

Return Values

None

isis_nsm_redistribute_unset

This call sends a redistribute unset message request to NSM.

Syntax

```
void
isis_nsm_redistribute_unset (struct isis_master *im, u_char proto, int type)
```

Input Parameters

`*im`	Represents the ISIS master structure.
`proto`	Defines IPv4/IPv6.
`type`	Message type.

Output Parameters

None

Return Values

None

nsm_server_set_callback

This call sends a redistribute unset message request to NSM.

Syntax

```
void
nsm_server_set_callback (struct nsm_server *ns, int message_type,
                         NSM_PARSER parser, NSM_CALLBACK callback)
```

Input Parameters

`*ns`	Represent the NSM server structure.
`parser`	Represents NSM parser.
`type`	Message type.

Output Parameters

None

Return Values

None

isis_nsm_recv_bgp_converge_done

This function is called when NSM receives information from BGP that convergence is complete and NSM calls the isis_overload_bit_wait_for_bgp_unset API in ISIS to unset the overload bit. The ISIS instance is checked and, if the overload bit is set, it is cleared. If the default timer expires, NSM calls isis_overload_unset and the `ISIS_OL_BIT_WAIT_FOR_BGP` flag is unset in that API call.

Syntax

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

Input Parameters

`*header`	Structure for the NSM message header.
`*arg`	Argument pointer.
`*message`	Message received.

Output Parameters

None

Return Values

0

isis_overload_set

This call sets the ISIS overload feature.

Syntax

```
void
isis_overload_set (struct isis *top)
```


Input Parameters

`*top`	Represents ISIS to structure.

Output Parameters

None

Return Values

0

isis_overload_unset

This call unsets the ISIS overload feature.

Syntax

```
void
isis_overload_unset (struct isis *top)
```

Input Parameters

`*top`	Represents ISIS to structure.

Output Parameters

None

Return Values

0

nsm_server_recv_bgp_conv_done

This function checks the BGP received flag to see if convergence is done.

Syntax

```
s_int32_t
nsm_server_recv_bgp_conv_done (struct nsm_msg_header *header,
                               void *arg, void *message)
```

Input Parameters

`*header`	Structure for the NSM message header.
`*arg`	Argument pointer.
`*message`	Message received.

Output Parameters

None

Return Values

0

bgp_check_peer_convergence

This function checks BGP to see if all peers have converged.

Syntax

```
void
bgp_check_peer_convergence (struct bgp *bgp)
```

Input Parameters

`*bgp`	Represents BGP main structure.

Output Parameters

None

Return Values

None

CHAPTER 6 Passive Interface

The passive interface feature for IS-IS lets the end user advertise a direct route without having a peer on that interface. A passive interface only advertises its IP address in its LSPs; it does not send or receive IS-IS packets.

Overview

Passive interface simplifies the configuration of distribution routers and allows network managers to obtain routing information from the interfaces in large ISP and enterprise networks.

The `passive-interface` command puts a specified interface, or all interfaces, into passive mode, except the high-priority interface. (There should be at least one IS-IS enabled interface present).

Priority is based on interface type and number. If the same type of interface is present, IS-IS checks for the highest interface number. For example, if there are 5 IS-IS enabled interfaces, loopback 0, loopback 1, eth0, eth1, eth2: if the `passive-interface` command is executed, all interfaces are put into passive mode, except loopback 1.

The `no` form of this command removes all interfaces, or the specified interface, from passive mode if they are already in passive mode.

To advertise networks through passive interfaces, one of the router's interfaces must be enabled with the `ip router isis` command.

Enabling passive interface on an IS-IS enabled interface disables IS-IS on the interface and makes the interface passive.

System Architecture

The following provides a configuration example, then describes configuration with passive-interface.

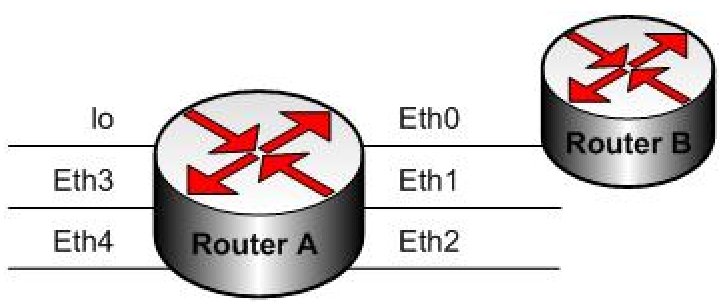

Figure 6-1: Passive Interface

In the figure above, Router A and Router B are connected with interface Eth0, and the other interfaces are connected to other networks.

To obtain routing information of all connected interfaces of Router A at Router B, set all interfaces as passive by default using a single `passive-interface` command, then configure individual interfaces where adjacencies are desired using the `no passive-interface` command with the interface.

CHAPTER 7 VLOG Support

This chapter discusses VLOG support in IS-IS.

Overview

VLOG provides a non-PVR (privileged virtual router) user the ability to debug on a VR, as well as allow these users to view debug information for the VR where the user is logged in. Conversely, VLOG provides a PVR user the ability to debug in the PVR context and other global debugging features not VR-specific. In addition, a PVR user may view all ZebOS-XP debugging, including debugging information generated in the context of non-privileged VRs. Finally, VLOG provides the ability to exclude log throttling (duplicate debug messages not handled) and user permissions for log file.

Features

- A VLOG build enables debugging in a specific VR context for BGP, OSPFv2, OSPFv3, IS-IS, RIP and RIPng.
- Commands entered in global VR configure mode allows a VR user to configure a system for a specific type of debugging.
- VR-specific debug output can be written to the terminals where a VR user has logged in.
- A PVR user may view all debug output, including those generated in the context of specific virtual routers.
- Debugging for protocol modules OSPFv2, ISIS, OSPFv3, RIPng, RIP and BGP is VR context-sensitive.

VLOG Users

There are two types of VLOG users, VR users and PVR users.

VR Users

VR users may enable or disable VR debugging and view the debug messages of its own VR. In addition, a VR user are able to log debug messages to a specified log file and specify a VR log file name.

PVR Users

PVR users may enable or disable PVR and VR (after logging in) debugging and view the PVR and any VR debug output from a PVR terminal session. In addition, a PVR user can log the debug output for both the PVR and all VRs to a log file and specify a local or global log file name.

VLOG Support in IS-IS

The following subsections describe the VLOG support.

Set Virtual Router Context

To make all VR debug commands context-specific, the following macro is called:

```
#define LIB_GLOB_SET_VR_CONTEXT(LIB_GLOB, VR_CXT) \
```

```
do {                                                   \
  ((LIB_GLOB)->vr_in_cxt) = (VR_CXT);                  \
} while (0)
```

This macro is defined in lib.h. It passes arguments in `lib_global (ZG)` and the pointer for binding VR from `isis_master`. This sets the VR context (`VR_CXT`) in lib_globals (`LIB_GLOB`). To set VR context for all internal or external events, it is necessary to identify specific events that can occur in the IS-IS system, and need to be modified or extended to support VR in IS-IS. When a specified event occurs, a determination is made as to whether a particular debug option is enabled in the isis_master database. If the debug option is enabled, the debug message (error, warning, informational) is displayed using either the `zlog_info`, `zlog_err` or `zlog_warn` function, which redirects the message to `VLOGD`. The VLOG module is fully responsible for displaying debug messages to the terminals or log files for VR-specific debugging.

Debug Flags Per Virtual Router

In IS-IS, the isis_master maintains system-wide configurations and variables. Debug flags for configuration and terminals are also maintained in the isis_master database. Each VR maintains an instance of the isis_master database, therefore, the debug flags enabled on one terminal or VR are distinguishable from another. The diagram depicts VR 01, VR 09 and a PVR. The context of each VR can be seen in its relationship to the isis_master database, with the debug flags set for it.

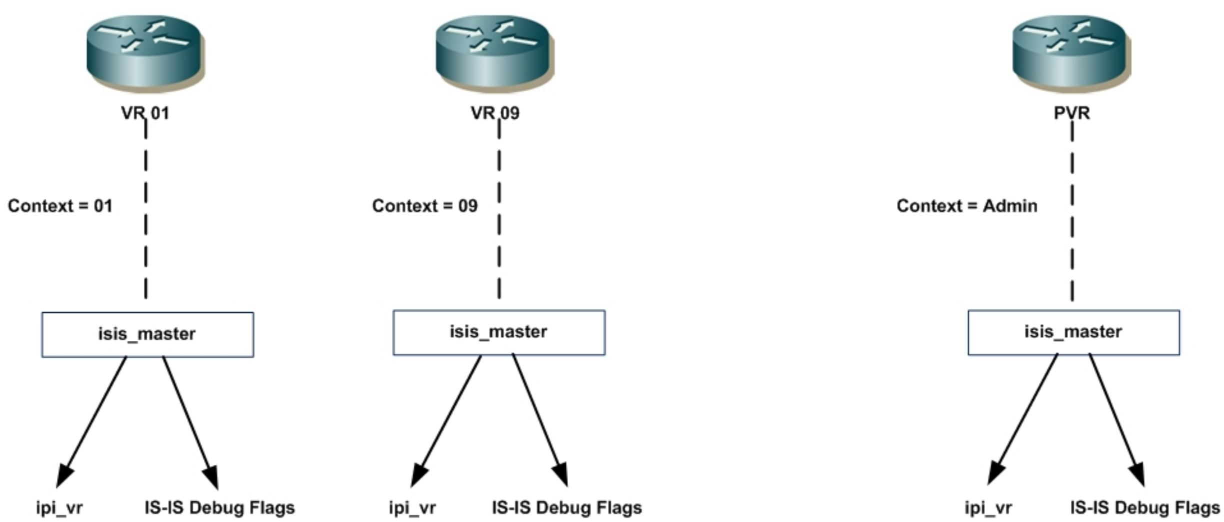

Figure 7-1: Virtual Routers in IS-IS Context

CHAPTER 8 Administrative Distance

Routers use administrative distance to select the best path when there are two or more different routes to the same destination from two different routing protocols. Administrative distance defines the reliability of a routing protocol. The smaller the administrative distance value, the more reliable the protocol. The following are the default administrative distance values for various protocols. The default IS-IS administrative distance value is 115.

Route Source	Default Distance Value
IGRP	100
OSPF	110
IS-IS	115
RIP	120

For example, if a router receives a route to a certain prefix from both IS-IS (default administrative distance – 115) and OSPF (default administrative distance -110), the router chooses the route learned through OSPF because OSPF has less administrative distance than IS-IS. If a user wants an IS-IS-learned route to be selected, instead of OSPF-learned routes to the same destination, the administrative distance for OSPF must be set to more than 115, or the administrative distance of IS-IS should be reduced to a value less than 100. The `distance` command is used for this purpose, so a user can configure the administrative distance for the protocol, thus altering the reliability of the protocol.

In the following diagram, R4 learns a route to the same prefix from OSPF, IS-IS and IGRP. Because the IGRP route has higher preference (with lower default administrative distance) than the OSPF and IS-IS routes, R4 prefers the IGRP route. To change the preference for OSPF and IS-IS routes, reduce the administrative distance for OSPF and IS-IS, or increase the IGRP administrative distance, using the `distance` command.

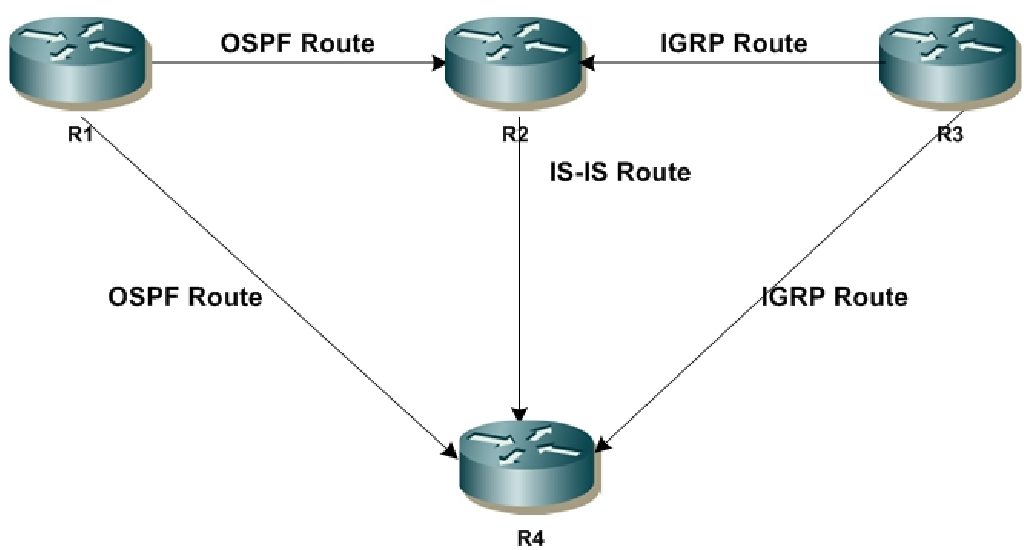

Figure 8-1: IS-IS Distance Configuration

The `distance` command can also be used to configure administrative distance for routes from a specific source, and for routes permitted by a particular access list. For example, in diagram above, administrative distance can be configured on R4 from IS-IS routes from R2. Also, if an access list is configured on R4, the `distance` command can be used to configure distance for all routes permitted by this access list.

System Architecture

NSM receives route information, along with administrative distance from various protocols. As shown in the diagram below, NSM receives route information from the IS-IS and OSPF protocols with the distance value set. These routes are stored in the NSM routing database.

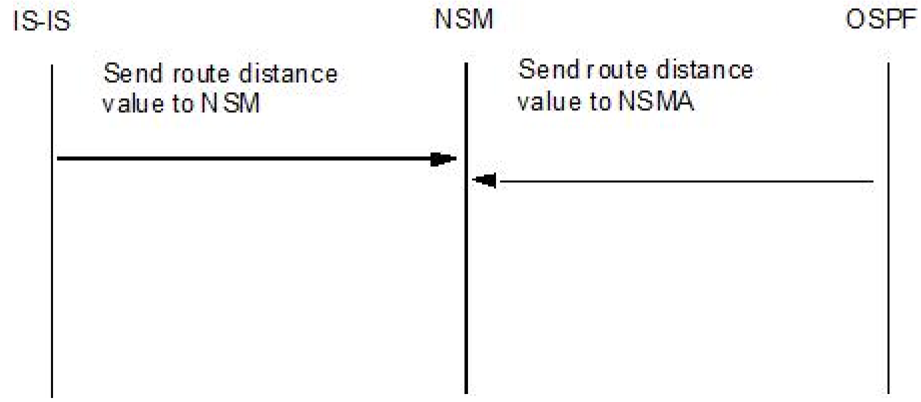

Figure 8-2: Distance Communication Flow

The routing table is derived from the routing database. Each entry in a routing table may specify a network: one destination address may match more than one routing table entry. The most specific table entry is selected, based on the longest prefix match. If these are multiple routes for the same distance, the route with least administrative distance is selected in the routing table.

CHAPTER 9 IS-IS Command API

This chapter contains Command Line Interface APIs for the IS-IS protocol.

Note: The `vr_id` parameter in each API supports a virtual router (VR). For an implementation without a VR, you must pass value 0 for the `vr_id` parameter.

Command Line Interface APIs

These two files contain the command line interface (CLI) and the application programming interface to it:

Module Name	Description
isis_api.c	Contains the functions that actually do the work of the commands in API format.
isis_cli.c	Contains the command definitions for the CLI that call the API functions.

isis_adjacency_check_ipv4_set

This call implements the `adjacency-check` command in the router mode. It enables adjacency check based on the IPv4 protocol TLVs in the IS-IS hello packet.

Syntax

```
int
isis_adjacency_check_ipv4_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_adjacency_check_ipv4_unset

This call implements the `no adjacency-check` command in the router mode. It disables adjacency check based on the IPv4 protocol TLVs in the IS-IS Hello packet.

Syntax

```
int
```

```
isis_adjacency_check_ipv4_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_adjacency_check_ipv6_set

This call implements the `adjacency-check` command in the address family IPv6 mode. It enables adjacency check based on the IPv6 protocol TLVs in the IS-IS Hello packet.

Syntax

```
int
isis_adjacency_check_ipv6_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_adjacency_check_ipv6_unset

This call implements the `no adjacency-check` command in the address family IPv6 mode. It disables adjacency check based on the IPv6 protocol TLVs in the IS-IS Hello packet.

Syntax

```
int
isis_adjacency_check_ipv6_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_metric_style_transition_set

This call implements the `metric-style transition` command to configure the metric-style transition in TLVs.

Syntax

```
int
isis_metric_style_transition_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_TE_ENABLED` if TE is enabled.

`ISIS_API_SET_ERR_MULTI_TOPOLOGY_ENABLED` if Multi topology is enabled.

`ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED` if protocol topology is enabled.

isis_metric_style_transition_narrow_set

This call implements the `metric-style transition narrow` command to configure metric-style as transition narrow in TLVs.

Syntax

```
int
isis_metric_style_transition_narrow_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_TE_ENABLED` if TE is enabled.

`ISIS_API_SET_ERR_MULTI_TOPOLOGY_ENABLED` if Multi topology is enabled.

`ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED` if protocol topology is enabled.

isis_metric_style_transition_wide_set

This call implements the `metric-style transition wide` command to configure metric-style as transition wide in TLVs.

Syntax

```
int
isis_metric_style_transition_wide_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2

3	Both Level-1, Level-2

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_INVALID_IS_TYPE when given level is outside of the range.

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_TE_ENABLED if TE is enabled.

ISIS_API_SET_ERR_MULTI_TOPOLOGY_ENABLED if Multi topology is enabled.

ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED if protocol topology is enabled.

isis_multi_topology_set

This call implements the `multi-topology` command to configure topology type as multi-topology in TLVs and SPF calculation.

Syntax

```
int
isis_multi_topology_set (u_int32_t vr_id, char *tag, u_char level)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
level	IS-IS instance level:
1	Level-1
2	Level-2
3	Both Level-1, Level-2

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_INVALID_IS_TYPE when given level is outside of the range.

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET if the Wide metric is not set.

ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED if protocol topology is enabled.

ISIS_API_SET_SUCCESS when the call is successful.

isis_multi_topology_transition_set

This call implements the `multi-topology transition` command to configure topology type as multi-topology transition in TLVs and SPF calculation.

Syntax

```
int
isis_multi_topology_transition_set (u_int32_t vr_id, char *tag, u_char level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET` if the Wide metric is not set.

`ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED` if protocol topology is enabled.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_multi_topology_unset

This call implements the `no multi-topology` command to configure the topology type as single-topology in TLVs and SPF calculation. Default is single-topology.

Syntax

```
int
isis_multi_topology_unset (u_int32_t vr_id, char *tag, u_char level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

isis_area_password_set

This call implements the `area-password` command to set authentication password for IS-IS L1 area. The configuration is stored regardless of whether IS-IS L1 instance is configured.

Syntax

```
int
isis_area_password_set (u_int32_t vr_id, char *tag, char *passwd);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`passwd`	Authentication key, null-terminated.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_PASSWORD_TOO_LONG` if length of password is longer than ISIS_MAX_PASSWD_LEN(254).

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_area_password_unset

The call implements the `no` parameter of the `area-password` command to unset authentication password for IS-IS L1 area.

Syntax

```
int
isis_area_password_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_l1_snp_auth_send_only

Syntax

```
int
isis_l1_snp_auth_send_only (u_int32_t vr_id, char *tag, char *passwd)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`*passwd`	Authentication key, null-terminated.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_PASSWORD_TOO_LONG` if length of password is longer than ISIS_MAX_PASSWD_LEN(254).

`ISIS_API_SET_ERR_AUTH_MD5_EXIST` if the level is already configured for MD5 authentication.

`ISIS_API_SET_ERR_AUTH_TEXT_EXIST`

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_l2_snp_auth_validate_set

Syntax

```
int
isis_l2_snp_auth_validate_set (u_int32_t vr_id, char *tag, char *passwd)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`*passwd`	Authentication key, null-terminated.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_PASSWORD_TOO_LONG` if length of password is longer than ISIS_MAX_PASSWD_LEN(254).

`ISIS_API_SET_ERR_AUTH_MD5_EXIST` if the level is already configured for MD5 authentication.

`ISIS_API_SET_ERR_AUTH_TEXT_EXIST`

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_l2_snp_auth_send_only

Syntax

```
int
isis_l2_snp_auth_send_only (u_int32_t vr_id, char *tag, char *passwd)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`*passwd`	Authentication key, null-terminated.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_PASSWORD_TOO_LONG` if length of password is longer than ISIS_MAX_PASSWD_LEN(254).

`ISIS_API_SET_ERR_AUTH_MD5_EXIST` if the level is already configured for MD5 authentication.

`ISIS_API_SET_ERR_AUTH_TEXT_EXIST`

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_ispf_set

This call sets ISIS iSPF.

Syntax

```
int
isis_ispf_set (u_int32_t vr_id, char *tag, int ispf_level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`ispf_level`	The iSPF level

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_ispf_unset

This call unsets ISIS iSPF.

Syntax

```
int
isis_ispf_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_lsp_mtu_set

This call sets the LSP MTU.

Syntax

```
int
isis_lsp_mtu_set (u_int32_t vr_id, char *tag, int size, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`size`	IS-IS size
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2

3	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_lsp_mtu_set

This call unsets the LSP MTU.

Syntax

```
int
isis_lsp_mtu_unset (u_int32_t vr_id, char *tag, int level)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
*tag	IS-IS instance area tag.
level	IS-IS instance level:
1	Level-1
2	Level-2
3	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_high_priority_tag_set

This call sets the high-priority tag.

Syntax

```
int
isis_high_priority_tag_set (u_int32_t vr_id, char *tag, u_int32_t priority_tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.
`priority_tag`	IS-IS priority tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_lsp_mtu_unset

This call unsets the high-priority tag.

Syntax

```
int
isis_high_priority_tag_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_if_tag_set

This call sets the priority tag

Syntax

```
int
isis_if_tag_set (u_int32_t vr_id, char *name,
                 u_int32_t tag, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*name`	Interface name

`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_if_tag_unset

This call unsets the priority tag

Syntax

```
int
isis_if_tag_unset (u_int32_t vr_id, char *name, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*name`	Interface name
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when the interface parameters are not configured.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_prc_interval_set

This call sets the parameters for PRC computation.

Syntax

```
int
isis_prc_interval_set (u_int32_t vr_id, char *tag,
                       u_int32_t min_delay, u_int32_t max_delay)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`min_delay`	Minimum delay between receiving a change to SPF calculation in milliseconds <0-2147483647>
`max_delay`	Maximum delay between receiving a change to SPF calculation in milliseconds <0-2147483647>

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_prc_interval_unset

This call resets the parameters for PRC computation.

Syntax

```
int
isis_prc_interval_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_proc_clear

Syntax

```
int
isis_proc_clear (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_clear_counters

Syntax

```
int
isis_clear_counters (u_int32_t vr_id)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_clear_interface_counters

Syntax

```
int
isis_clear_interface_counters (u_int32_t vr_id, char *ifname)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*ifname`	Interface name

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_clear_ip_route

Syntax

```
int
isis_clear_ip_route (u_int32_t vr_id, char *tag, char *str)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`*str`	

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_clear_ip_isis_route

Syntax

```
int
isis_clear_ip_isis_route (u_int32_t vr_id, char *tag, char *str)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`*str`	

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERROR` when LDP session state query was not successfully sent

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_cspf_set

This call implements the `capability-cspf` command to activate the CSPF feature in IS-IS.

Syntax

```
int
isis_cspf_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` for non-valid IS type.

`ISIS_API_SET_ERR_CSPF_ENABLE_FAILED` if the call failed for other reasons.

`ISIS_API_SET_SUCCESS` if the call is successful.

isis_cspf_unset

This call implements the `no` parameter of the `capability-cspf` command to deactivate the CSPF feature in IS-IS.

Syntax

```
int
isis_cspf_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if the ISIS instance does not exist.

ISIS_API_SET_ERR_CSPF_DISABLE_FAILED if the call failed for other reasons.

ISIS_API_SET_SUCCESS if the call is successful.

isis_default_information_originate_ipv4_set

This call implements the default-information originate command to inject IPv4 default route into IS-IS.

Syntax

```
int isis_default_information_originate_ipv4_set (u_int32_t vr_id, char *tag, char
*rmap_name);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
rmap_name	Name of route-map.

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist

ISIS_API_SET_SUCCESS if the call is successful

isis_default_information_originate_ipv4_unset

This call implements the no parameter of the default-information originate command to stop injecting the IPv4 default route into IS-IS.

Syntax

```
int isis_default_information_originate_ipv4_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist

ISIS_API_SET_SUCCESS if the call is successful

isis_default_information_originate_ipv6_set

This call implements the `default-information originate` command to inject the IPv6 default route into IS-IS.

Syntax

```
int isis_default_information_originate_ipv6_set (u_int32_t vr_id, char *tag, char
*rmap_name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`rmap_name`	Name of route-map.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_default_information_originate_ipv6_unset

This call implements the `no` parameter of the `default-information originate` command to stop injecting the IPv6 default route into IS-IS.

Syntax

```
int isis_default_information_originate_ipv6_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_distance_set

This call implements the `distance` command to define an administrative distance for all routes from a specific source and/or all routes permitted by an access-list.

Syntax

```
int isis_distance_source_set (u_int32_t vr_id, char *tag, u_int32_t distance,
char *sys_id, char *access_name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`sys_id`	Source ID.
`access_name`	Access-list name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_DISTANCE_INVALID` if the distance is not within the range.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_distance_unset

This call implements the `no distance` command to remove an administrative distance for all routes from a specific source and/or all routes permitted by an access-list.

Syntax

```
int
isis_distance_source_unset (u_int32_t vr_id, char *tag, char *sys_id,
                            char *access_name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`sys_id`	Source ID.
`access_name`	Access-list name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_DISTANCE_NOT_EXIST` if the distance does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_distance_source_set

This call implements the `distance` command to define an administrative distance for all routes from a specific route source and/or all routes permitted by an access-list.

Syntax

```
int
isis_distance_source_set (u_int32_t vr_id, char *tag, u_int32_t distance,
                          char *sys_id, char *access_name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`sys_id`	Source ID.
`access_name`	Access-list name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_DISTANCE_INVALID` if the distance is not within the range.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_distance_source_unset

This call implements the `no distance` command to remove an administrative distance for all a specific routes from a specific source and/or all routes permitted by an access-list.

Syntax

```
int
isis_distance_source_unset (u_int32_t vr_id, char *tag, char *sys_id, char
*access_name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`sys_id`	Source ID.
`access_name`	Access-list name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_DISTANCE_NOT_EXIST if the distance does not exist.

ISIS_API_SET_SUCCESS when it is successful.

isis_distance_ipv6_set

This call implements the distance command, which defines an administrative distance for all routes for an IPv6 address family.

Syntax

```
int isis_distance_ipv6_set (u_int32_t vr_id, char *tag, u_int32_t distance)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
distance	Administrative distance.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_DISTANCE_INVALID if the distance is not within the range.

ISIS_API_SET_SUCCESS when it is successful.

isis_distance_ipv6_unset

This call implements the no distance command to remove an administrative distance for all routes for an IPv6 address family.

Syntax

```
int isis_distance_ipv6_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_DISTANCE_NOT_EXIST if the distance does not exist.

ISIS_API_SET_SUCCESS when it is successful.

isis_domain_password_set

This call implements the domain-password command to set the authentication password for the IS-IS L2 routing domain. The configuration is stored regardless of whether the IS-IS L2 instance is configured.

Syntax

```
int isis_domain_password_set (u_int32_t vr_id, char *tag, char *passwd);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
passwd	Authentication key, null-terminated.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_ERR_PASSWORD_TOO_LONG if length of password is longer than ISIS_MAX_PASSWD_LEN(254).

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_SUCCESS when the call is successful.

isis_domain_password_unset

The call implements the no parameter of the domain-password command to unset the authentication password for the IS-IS L2 routing domain.

Syntax

```
int isis_domain_password_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_SUCCESS when the call is successful.

isis_hostname_dynamic_set

This call implements the `dynamic-hostname` command to configure the dynamic hostname TLV capability.

Syntax

```
int isis_hostname_dynamic_set (u_int32_t vr_id, char *tag, int flag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`flag`	Method for dynamic-hostname:
`0`	Hostname given by router `hostname' command.
`1`	Hostname given by IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_hostname_dynamic_unset

This call implements the `no` parameter of the `dynamic-hostname` command to unconfigure dynamic-hostname TLV capability.

Syntax

```
int isis_hostname_dynamic_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_circuit_type_set

This call implements the `isis circuit-type` command to change the interface's circuit type.

Syntax

```
int isis_if_circuit_type_set (u_int32_t vr_id, char *name, int type);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`*name`	IS-IS instance area tag.
`type`	IS-IS Circuit-type:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given is-type is outside of the range.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` if interface of given name is not enabled for IS-IS yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_circuit_type_unset

This call implements the `no` parameter of the `isis circuit-type` command to change the interface's circuit type to the default.

Syntax

```
int isis_if_circuit_type_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist

`ISIS_API_SET_ERR_IF_NOT_ENABLED` if interface of given name is not enabled for IS-IS yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_csnp_interval_set

This call implements the `isis csnp-interval` command to configure the interface's CSNP interval.

Syntax

```
int isis_if_csnp_interval_set (u_int32_t vr_id, char *name, u_int32_t interval, int
level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`interval`	Interval in seconds. <0-65535>.
`level`	IS-IS instance level:
1	Level-1
2	Level-2
3	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_csnp_interval_unset

This call implements the `no` parameter of the `isis csnp-interval` command to unconfigure the interface's CSNP interval. Default is 10 (seconds).

Syntax

```
int isis_if_csnp_interval_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`level`	IS-IS instance level:
1	Level-1
2	Level-2
3	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_interval_set

This call implements the `isis hello-interval` command to configure interface's Hello interval.

Syntax

```
int isis_if_hello_interval_set (u_int32_t vr_id, char *name, u_int32_t interval, int
level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`interval`	Interval in seconds. <0-65535>.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_interval_minimal_set

This call implements the `isis hello-interval minimal` command to configure the Holdtime in Hello PDU to 1 second.

Syntax

```
int isis_if_hello_interval_minimal_set (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`level`	IS-IS instance level:

`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_interval_unset

This call implements the `no` parameter of the `isis hello-interval` command to unconfigure the interface's Hello interval. Default is 10 (seconds).

Syntax

```
int isis_if_hello_interval_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`interval`	Interval in seconds. <0-65535>.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_multiplier_set

This call implements the `isis hello-multiplier` command to configure the interface's Hello-Multiplier value.

Syntax

```
int
```



```
isis_if_hello_multiplier_set (u_int32_t vr_id, char *name,
                              u_int32_t multi, int level)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
name	Interface name
multi	Multiplier for Hello holding time. <2-100>
level	IS-IS instance level:
1	Level-1
2	Level-2
3	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given multiplier is outside of the range.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_multiplier_unset

This call implements the `no` parameter of the `isis hello-multiplier` command to unconfigure the interface's Hello-Multiplier value. Default is 3.

Syntax

```
int isis_if_hello_multiplier_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
name	Interface name
level	IS-IS instance level:
1	Level-1
2	Level-2
3	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_hello_padding_set

This call implements the `isis hello padding` command. It enables IS-IS Hello packet padding.

Syntax

```
int isis_if_hello_padding_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_if_hello_padding_unset

This call implements the `no isis hello padding` command. It disables IS-IS Hello packet padding.

Syntax

```
int isis_if_hello_padding_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when it is successful.

isis_if_ip_router_set

This call implements the `ip router isis` command to enable IP router interface commands.

Syntax

```
int isis_if_ip_router_set (u_int32_t vr_id, char *name, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if length of area tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_AREA_TAG_NOT_MATCHED` if given area tag is not matched to configured one.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_ip_router_unset

This call implements the `no` parameter of the `ip router isis` command to disable IP router interface commands.

Syntax

```
int isis_if_ip_router_unset (u_int32_t vr_id, char *name, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if length of area tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_AREA_TAG_NOT_MATCHED` if given area tag is not matched to configured one.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_ipv6_router_set

This call implements the `ipv6 router isis` command to enable the interface for IPv6 routing.

Syntax

```
int isis_if_ipv6_router_set (u_int32_t vr_id, char *name, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if given tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_AEEA_TAG_NOT_MATCHED` if instance exists and given tag does not match to existing one.

`ISIS_API_SET_ERR_IF_NOT_EXIST` if interface does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_ipv6_router_unset

This call implements the `no` parameter of the `ipv6 router isis` command to disable the interface for IPv6 routing.

Syntax

```
int isis_if_ipv6_router_unset (u_int32_t vr_id, char *name, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if given tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_AEEA_TAG_NOT_MATCHED` if instance exists and given tag does not match to existing one.

`ISIS_API_SET_ERR_IF_NOT_EXIST` if interface does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_level_conf_ldp_igp_sync

This call enables LDP IS-IS synchronization on the interface.

Syntax

```
bool_t
isis_if_level_conf_ldp_igp_sync (u_int32_t vr_id,char *name,int level,
                                 u_int32_t holddown_timer)
```

Input Parameters

`vr_id`	Virtual router ID
`name`	Name of the interface on which LDP-ISIS synchronization is enabled
`level`	IS-IS level at which synchronization is enabled
`holddown_timer`	Sets the hold-down timer for synchronization

Output Parameters

None

Return Values

ISIS_API_SET_SUCCESS when call is successful

ISIS_API_SET_ERR_VR_NOT_EXIST when IS-IS master is not present

ISIS_API_SET_ERR_VR_NOT_EXIST when the interface does not exist

ISIS_API_SET_ERR_LEVEL_CHANGED When synchronization is re-configured at a different level.

ISIS_API_SET_ERR_LDP_IGP_SYNC_ENABLED when synchronization is re-configured with the same hold-down timer

ISIS_API_SET_ERROR when LDP session state query was not successfully sent

isis_if_level_conf_ldp_igp_unsync

This call disables LDP IS-IS synchronization on the interface.

Syntax

```
bool_t
isis_if_level_conf_ldp_igp_unsync (u_int32_t vr_id,char *name)
```

Input Parameters

`vr_id`	Virtual router ID
`name`	Name of the interface on which LDP-ISIS synchronization is enabled

Output Parameters

None

Return Values

PAL_TRUE when synchronization is successfully disabled

PAL_FALSE when synchronization is not successfully disabled

isis_if_lsp_interval_set

This call implements the `isis lsp-interval` command to configure the interface's LSP transmission interval.

Syntax

```
int isis_if_lsp_interval_set (u_int32_t vr_id, char *name, u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`interval`	Interval in milliseconds. <1-4294967295>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

iisis_if_lsp_interval_unset

This call implements the `no` parameter of the `isis lsp-interval` command to unconfigure interface's LSP transmission interval. Default is 33 (milliseconds).

Syntax

```
int isis_if_lsp_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_mesh_group_block_set

This call implements the `isis mesh-group blocked` command to configure the interface as mesh-group blocked.

Syntax

```
int isis_if_mesh_group_block_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_NOT_EXIST` if interface does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_SUCCESS` when the call is successful

isis_if_mesh_group_set

This call implements the `isis mesh-group` command to configure the mesh group ID.

Syntax

```
int isis_if_mesh_group_set (u_int32_t vr_id, char *name, u_int32_t group_id);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`group_id`	Mesh group ID.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_NOT_EXIST` if interface does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_SET_ERR_MESH_GROUP_ID_INVALID` when the mesh group ID is invalid

`ISIS_API_SET_SUCCESS` when the call is successful

isis_if_mesh_group_unset

This call implements the `no` parameter of the `isis mesh-group` command to unconfigure the mesh group ID or mesh group blocked.

Syntax

```
int isis_if_mesh_group_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_NOT_EXIST` if interface does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` when interface is not enabled for ISIS.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful

isis_if_metric_set

This call implements the `isis metric` command to configure the interface's metric value.

Syntax

```
int isis_if_metric_set (u_int32_t vr_id, char *name, u_char metric, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`metric`	Metric value.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given metric is outside of the range.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_metric_unset

This call implements the `no` parameter of the `isis metric` command to unconfigure the interface's metric value. Default is 10.

Syntax

```
int isis_if_metric_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_network_type_set

This call changes the IS-IS network type to the specified type.

Syntax

```
int isis_if_network_type_set (u_int32_t vr_id, char *name, int type);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`type`	Interface network type:

`ISIS_IFTYPE_POINTTOPOINT`

`ISIS_IFTYPE_BROADCAST`

Output Parameters

None

Return Value

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when the specified interface does not exist.

`ISIS_API_SET_ERR_INVALID_NETWORK_TYPE` when the specified network type is invalid for this interface.

isis_if_network_type_unset

This call changes the IS-IS network type to the default value.

Syntax

```
int isis_if_network_type_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.

Output Parameters

None

Return Value

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when the specified interface does not exist.

`ISIS_API_SET_ERR_INVALID_NETWORK_TYPE` when the specified network type is invalid for this interface.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not yet configured.

isis_if_password_set

This call implements the `isis password` command to configure the interface's authentication password.

Syntax

```
int isis_if_password_set (u_int32_t vr_id, char *name, char *passwd, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`passwd`	Authentication key, null-terminated.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_PASSWORD_TOO_LONG` if length of passwd is longer than ISIS_MAX_PASSWD_LEN(254).

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_password_unset

This call implements the `no` parameter of the `isis password` command to unconfigure the interface's authentication password.

Syntax

```
int isis_if_password_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_priority_set

This call implements the `isis priority` command to configure the interface's Priority value for Designated Router election.

Syntax

```
int isis_if_priority_set (u_int32_t vr_id, char *name, u_char priority, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`priority`	Priority for Designated Router election. <0-127>
`level`	IS-IS instance level:

`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given priority is outside of the range.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_priority_unset

This call implements the `no` parameter of the `isis priority` command to unconfigure the interface's Priority value. Default is 64.

Syntax

```
int isis_if_priority_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_retransmit_interval_set

This call implements the `isis retransmit-interval` command to configure the LSP retransmission interval.

Syntax

```
int isis_if_retransmit_interval_set (u_int32_t vr_id, char *name, u_int32_t interval);
```


Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`interval`	Interval value in seconds. <0-65535>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` if interface of given name is not enabled for IS-IS yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_retransmit_interval_unset

This call implements the `no` parameter of the `isis retransmit-interval` command to unconfigure the LSP retransmission interval.

Syntax

```
int isis_if_retransmit_interval_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_IF_NOT_ENABLED` if interface of given name is not enabled for IS-IS yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_wide_metric_set

This call implements the `isis wide-metric` command to configure the wide metric value for the interface.

Syntax

```
int isis_if_wide_metric_set (u_int32_t vr_id, char *name, u_int32_t metric, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`metric`	IS-IS metric value.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_VALUE` if given metric is outside of the range

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_wide_metric_unset

This call implements the `no` parameter of the `isis wide-metric` command to unconfigure the wide metric value for the interface.

Syntax

```
int isis_if_wide_metric_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Level-1-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside of the range.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` if no interface parameter is configured.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_ignore_lsp_errors_set

This call implements the `ignore-lsp-errors` command to ignore receiving LSPs with checksum error. LSP will be accepted as if it is valid.

Syntax

```
int isis_ignore_lsp_errors_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_ignore_lsp_errors_unset

This call implements the `no` parameter of the `ignore-lsp-errors` command to validate receiving the LSP checksum. The LSP will be rejected if the checksum has an error.

Syntax

```
int isis_ignore_lsp_errors_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_instance_set

This call implements the `router isis` command to create an IS-IS instance for enabling a routing process.

Syntax

```
int isis_instance_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	IS-IS instance area name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if length of area tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_INVALID_VALUE` when the instance is not valid.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_instance_unset

This call implements the `no` parameter of the `router isis` command to delete an IS-IS instance.

Syntax

```
int isis_instance_unset (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	IS-IS instance area name.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_AREA_TAG_TOO_LONG` if length of area tag is longer than ISIS_AREA_TAG_MAX_LEN(60).

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_is_type_set

This call implements the `is-type` command to change the level of the IS-IS instance. Because only one L2 instance can be configured, it is not allowed to change the level to L1 or L1L2 if L2 or L1L2 if the instance already exists.

Syntax

```
int isis_is_type_set (u_int32_t vr_id, char *tag, int is_type);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	IS-IS instance area name.

`is_type`	IS-IS instance level type:
`1`	Level-1
`2`	Level-2-only
`3`	Level-1-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given is-type is outside the range.

`ISIS_API_SET_ERR_L2_INSTANCE_EXIST` if L2 or L1L2 instance already exists.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_is_type_unset

This call implements the `no` parameter of the `is-type` command to reset IS-IS Level to default. Default is L1L2. If L2 instance already exists, the configuration cannot be unconfigured.

Syntax

```
int isis_is_type_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_L2_INSTANCE_EXIST` if L2 or L1L2 instance already exists.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_lsp_gen_interval_set

This call implements the `lsp-gen-interval` command to configure the minimum interval between regenerating the same LSP.

Syntax

```
int isis_lsp_gen_interval_set (u_int32_t vr_id, char *tag, int level, u_char interval);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	IS-IS instance area name.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2
`interval`	Interval in seconds. <1-120>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_lsp_gen_interval_unset

This call implements the `no` parameter of the `lsp-gen-interval` command to unconfigure the minimum interval between regenerating the same LSP. Default value is 30 (seconds).

Syntax

```
int isis_lsp_gen_interval_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_lsp_refresh_interval_set

This call implements the `lsp-refresh-interval` command to configure the LSP refresh interval.

Syntax

```
int isis_lsp_refresh_interval_set (u_int32_t vr_id, char *tag, u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`interval`	Interval in seconds. <1-65535>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_lsp_refresh_interval_unset

This call implements the `no` parameter of the `lsp-refresh-interval` command to unconfigure the LSP refresh interval. Default value is 900 (seconds).

Syntax

```
int isis_lsp_refresh_interval_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_max_area_addr_set

This call implements the `max-area-address` command to set the maximum number of ISIS areas that can be configured on a router with the `net` command. By default, ISIS permits a maximum of three areas that can be defined on a router.

Syntax

```
int
```

```
isis_max_area_addr_set (u_int32_t vr_id, char *tag,
                            u_char limit)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`limit`	The maximum number of areas in the network `<3-254>`.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_VALUE` when the given limit is outside of the range.

`ISIS_API_SET_ERR_MAX_AREA` when the current number of areas exceeds the given limit.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_max_area_addr_unset

This call implements the `no max-area-address` command to set the maximum number of ISIS areas to its default (3).

Syntax

```
int
isis_max_area_addr_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_max_lsp_lifetime_set

This call implements the `max-lsp-lifetime` command to configure the maximum LSP lifetime.

Syntax

```
int isis_max_lsp_lifetime_set (u_int32_t vr_id, char *tag, u_int32_t max_lifetime);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`max_lifetime`	Maximum LSP lifetime in seconds. <1-65535>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_VALUE` when given max_lifetime is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_max_lsp_lifetime_unset

This call implements the `no` parameter of the `max-lsp-lifetime` command to unconfigure the maximum LSP lifetime, and set it to the default value 1200 (seconds).

Syntax

```
int isis_max_lsp_lifetime_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_metric_style_set

This call implements the `metric-style` command to configure the metric style as wide in TLVs.

Syntax

```
int isis_metric_style_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_TE_ENABLED` when ISIS TE is enabled.

`ISIS_API_SET_ERR_MULTI_TOPOLOGY_ENABLED` when Multi Topology is enabled.

`ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED` when Protocol Topology is enabled.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_metric_style_unset

This call implements the `no` parameter of the `metric-style` command to unconfigure the metric style in TLVs. Default is narrow.

Syntax

```
int isis_metric_style_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

ISIS_API_SET_ERR_TE_ENABLED when given level is configured as TE-enabled. As a side-effect, it also removes MPLS traffic-engineering configuration of this level.

ISIS_API_SET_ERR_MULTI_TOPOLOGY_ENABLED when Multi Topology is enabled.

ISIS_API_SET_ERR_PROTOCOL_TOPOLOGY_ENABLED when Protocol Topology is enabled.

isis_mpls_traffic_eng_router_id_set

This call implements the mpls traffic-eng router-id command to configure the TE router-ID.

Syntax

```
int isis_mpls_traffic_eng_router_id_set (u_int32_t vr_id, char *tag, struct
pal_in4_addr router_id);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
router_id	Router ID to be set.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_SUCCESS when the call is successful.

isis_mpls_traffic_eng_router_id_unset

This call implements the no parameter of the mpls traffic-eng router-id command to unconfigure the TE router-ID.

Syntax

```
int isis_mpls_traffic_eng_router_id_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist with given tag.

ISIS_API_SET_SUCCESS when the call is successful.

isis_mpls_traffic_eng_set

This call implements the `mpls traffic-eng` command to enable IS-IS TE extension.

Syntax

```
int isis_mpls_traffic_eng_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_TE_ENABLED` if the other level is already configured as TE-enabled.

`ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET` when wide-metric is not configured.

isis_mpls_traffic_eng_unset

This call implements the `no` parameter of the `mpls traffic-eng` command to disable IS-IS TE extension.

Syntax

```
int isis_mpls_traffic_eng_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_TE_NOT_ENABLED` if given level is not configured as TE-enabled.

isis_net_set

This call implements the `net` command to configure Network Entity Title (NET) for this process.

Syntax

```
int isis_net_set (u_int32_t vr_id, char *tag, char *net);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`net`	Network entity title in string.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_NET_WRONG_FORMAT` if the format of NET is invalid.

`ISIS_API_SET_ERR_NET_INVALID_LENGTH` if the length of area address is less than 8 octets or more than 20 octets.

`ISIS_API_SET_ERR_SYSTEM_ID_CANT_CHANGED` if system ID is not matched to other NETs.

`ISIS_API_SET_ERR_TOO_MANY_AREA_ADDRESSES` if maximum number of area address are already configured.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_net_unset

This call implements the `no` parameter of the `net` command to unconfigure the Network Entity Title.

Syntax

```
int isis_net_unset (u_int32_t vr_id, char *tag, char *net);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`net`	Network entity title in string.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_NET_WRONG_FORMAT` if the format of NET is invalid.

`ISIS_API_SET_ERR_NET_INVALID_LENGTH` if the length of area address is less than 8 octets, or more than 20 octets.

`ISIS_API_SET_ERR_SYSTEM_ID_NOT_CONFIGURED` if no system ID nor area address is configured yet.

`ISIS_API_SET_ERR_SYSTEM_ID_NOT_MATHCED` if system ID is not matched to existing NETs.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_protocol_topology_set

This call implements the `protocol-topology` command to enable Protocol Topology support.

Syntax

```
int isis_protocol_topology_set (u_int32_t vr_id, char *tag, u_char level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2-only
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET` when the metric type is not configured as wide.

`ISIS_API_SET_ERR_MULTI_TOPOLOGY_SET` when Multi topology is set.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_protocol_topology_unset

This call implements the `no` parameter of the `protocol-topology` command to disable Protocol Topology support.

Syntax

```
int isis_protocol_topology_unset (u_int32_t vr_id, char *tag, u_char level);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
level	IS-IS instance level:
1	Level-1
2	Level-2-only
3	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_redistribute_inter_level_ipv4_set

This call implements the `redistribute isis` command for IPv4 to configure inter-level redistribution.

Syntax

```
int isis_redistribute_inter_level_ipv4_set (u_int32_t vr_id, char *tag, int level, char
*name);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.
level	IS-IS instance level:
1	Level-1
2	Level-2
name	Access-list name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_redistribute_inter_level_ipv4_unset

This call implements the no parameter of the `redistribute isis` command for IPv4 to unconfigure inter-level redistribution.

Syntax

```
int isis_redistribute_inter_level_ipv4_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_redistribute_inter_level_ipv6_set

This call implements the `redistribute isis` command for IPv6 to configure inter-level redistribution.

Syntax

```
int isis_redistribute_inter_level_ipv6_set (u_int32_t vr_id, char *tag, int level, char
*name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`name`	Access-list name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_redistribute_inter_level_ipv6_unset

This call implements the `no redistribute isis` command for IPv6 to unconfigure inter-level redistribution.

Syntax

```
int isis_redistribute_inter_level_ipv6_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_redistribute_ipv4_set

This call implements the `redistribute` command to inject IPv4 routes into IS-IS from another routing protocol.

Syntax

```
int isis_redistribute_ipv4_set (u_int32_t vr_id, char *tag, int source, u_int32_t
metric, u_char metric_type, int level, char *rmap_name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`source`	Source of protocol:
`1`	Kernel routes
`2`	Connected routes

3	Static routes
4	RIP routes
5	OSPF routes
6	BGP routes
metric	IS-IS metric
metric_type	External metric type:
1	Internal
2	External
level	IS-IS instance level:
1	Level-1
2	Level-2
3	Both Level-1, Level-2
rmap_name	Name of route-map.

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the VR does not exist.

ISIS_API_SET_ERR_INVALID_ROUTE_TYPE if given route type is outside of the range

ISIS_API_SET_ERR_INVALID_METRIC_VALUE if given metric value is outside of the range

ISIS_API_SET_ERR_INVALID_METRIC_TYPE if given metric type is outside of the range

ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET if given metric value is greater than 63.

ISIS_API_SET_ERR_INVALID_IS_TYPE if given level is outside of the range

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if IS-IS instance does not exist

ISIS_API_SET_SUCCESS if the call is successful

isis_redistribute_ipv4_unset

This call implements the `no` parameter of the `redistribute` command to stop injecting IPv4 routes into IS-IS from another routing protocol.

Syntax

```
int isis_redistribute_ipv4_unset (u_int32_t vr_id, char *tag, int source);
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance tag
source	Source of protocol:
1	Kernel routes
2	Connected routes

`3`	Static routes
`4`	RIP routes
`5`	OSPF routes
`6`	BGP routes

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_ROUTE_TYPE` if given route type is outside of the range

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_redistribute_ipv6_set

This call implements the `redistribute` command to inject IPv6 routes into IS-IS from other routing protocol.

Syntax

```
int isis_redistribute_ipv6_set (u_int32_t vr_id, char *tag, int source, u_int32_t
metric, u_char metric_type, int level, char *rmap_name);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`source`	Source of protocol:
`1`	Kernel routes
`2`	Connected routes
`3`	Static routes
`4`	RIPng routes
`5`	OSPFv3 routes
`6`	BGP4+ routes
`metric`	IS-IS metric
`metric_type`	External metric type:
`1`	Internal
`2`	External
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2
`rmap_name`	Name of route-map.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_ROUTE_TYPE` if given route type is outside of the range

`ISIS_API_SET_ERR_INVALID_METRIC_VALUE` if given metric value is outside of the range

`ISIS_API_SET_ERR_INVALID_METRIC_TYPE` if given metric type is outside of the range

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside of the range

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_redistribute_ipv6_unset

This call implements the `no` parameter of the `redistribute` command to stop injecting IPv6 routes into IS-IS from another routing protocol.

Syntax

```
int isis_redistribute_ipv6_unset (u_int32_t vr_id, char *tag, int source);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`source`	Source of protocol:
`1`	Kernel routes
`2`	Connected routes
`3`	Static routes
`4`	RIPng routes
`5`	OSPFv3 routes
`6`	BGP4+ routes

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_ROUTE_TYPE` if given route type is outside of the range

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_spf_interval_set

This call implements the `spf-interval-exp` command to configure the minimum and maximum interval between SPF calculations.

Syntax

```
int isis_spf_interval_set (u_int32_t vr_id, char *tag, int level, u_int32_t min_delay,
u_int32_t max_delay);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2
`min_delay`	Minimum delay between receiving a change to SPF calculation in milliseconds <0-2147483647>
`max_delay`	Maximum delay between receiving a change to SPF calculation in milliseconds <0-2147483647>

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_INVALID_VALUE` when given interval is outside of the range.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_spf_interval_unset

This call implements the `no` parameter of the `spf-interval` command to unconfigure the minimum interval between SPF calculations. Default is 10 (seconds).

Syntax

```
int isis_spf_interval_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_summary_address_set

This call implements the `summary-address` command to summarize specific IPv4 reachability information.

Syntax

```
int isis_summary_address_set (u_int32_t vr_id, char *tag, struct pal_in4_addr addr,
u_char masklen, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`addr`	IPv4 network address.
`masklen`	Mask length.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside of the range

`ISIS_API_SET_SUCCESS` if the call is successful

isis_summary_address_unset

This call implements the `no` parameter of the `summary-address` command to remove the summary.

Syntax

```
int isis_summary_address_unset (u_int32_t vr_id, char *tag, struct pal_in4_addr addr,
u_char masklen);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`addr`	IPv4 network address.

`masklen`	Mask length.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_summary_prefix_set

This call implements the `summary-prefix` command to summarize specific IPv6 reachability information.

Syntax

```
int isis_summary_prefix_set (u_int32_t vr_id, char *tag, struct pal_in6_addr addr,
u_char masklen, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`addr`	IPv6 network address.
`masklen`	Mask length.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_ERR_INVALID_IS_TYPE` if given level is outside of the range

`ISIS_API_SET_SUCCESS` if the call is successful

isis_summary_prefix_unset

This call implements the `no` parameter of the `summary-prefix` command to remove the summary.

Syntax

```
int isis_summary_prefix_unset (u_int32_t vr_id, char *tag, struct pal_in6_addr addr,
u_char masklen);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`addr`	IPv6 network address.
`masklen`	Mask length.

Output Parameters

None

Return Values

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist

`ISIS_API_SET_SUCCESS` if the call is successful

isis_auth_send_only_set

This call implements the `authentication send-only` command to configure the send-only option, that is, not to validate the authentication on the received LSP/CSNP/PSNP packets.

Syntax

```
int isis_auth_send_only_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`addr`	IPv4 network address.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_send_only_unset

This call implements the `no` parameter of the `authentication send-only` command to unconfigure the send-only option, that is, to validate the authentication on the received LSP/CSNP/PSNP packets.

Syntax

```
int isis_auth_send_only_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance tag
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_mode_hmac_md5_set

This call implements the `authentication mode md5` command to set the authentication mode to MD5.

Syntax

```
int isis_auth_mode_hmac_md5_set (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS area instance tag
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_mode_hmac_md5_unset

This call implements the `no` parameter of the `authentication mode md5` command to unset the authentication mode to MD5.

Syntax

```
int isis_auth_mode_hmac_md5_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS area instance tag
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_key_chain_set

This call implements the `authentication key-chain` command to set the key chain to be used for authentication on LSP/CSNP/PSNP packets.

Syntax

```
int isis_auth_key_chain_set (u_int32_t vr_id, char *tag, char *key_chain, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS area instance tag
`key_chain`	Key chain used for authentication.
`level`	IS-IS instance level:

1	Level-1
2	Level-2
3	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_NO_AUTH_MD5_EXIST` if the authentication mode is not set to MD5 for the level specified.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_key_chain_unset

This call implements the `no` parameter of the `authentication key-chain` command to remove the configured authentication key-chain.

Syntax

```
int isis_auth_key_chain_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS area instance tag
`level`	IS-IS instance level:
1	Level-1
2	Level-2
3	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_mode_text_set

This call implements the `isis authentication mode text` command to set the authentication mode to text.

Syntax

```
int
isis_auth_mode_text_set (u_int32_t vr_id, char *tag, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_ERR_AUTH_MD5_EXIST` if the level is already configured for MD5 authentication.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_auth_mode_text_unset

This call implements the `no` form of the `isis authentication mode text` command to unset text authentication mode.

Syntax

```
int
isis_auth_mode_text_unset (u_int32_t vr_id, char *tag, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_send_only_set

This call implements the `isis authentication send-only` command to configure the send-only option, that is, not to validate the authentication on the hello PDUs.

Syntax

```
int isis_if_auth_send_only_set (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_send_only_unset

This call implements the `no` form of the `isis authentication send-only` command to unconfigure the send-only option, that is, to validate the authentication on the hello PDUs.

Syntax

```
int isis_if_auth_send_only_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_mode_hmac_md5_set

This call implements the `isis authentication mode md5` command to set the authentication mode to MD5.

Syntax

```
int isis_if_auth_mode_hmac_md5_set (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_mode_hmac_md5_unset

This call implements the `no` parameter of the `isis authentication mode md5` command to unset the authentication mode to MD5.

Syntax

```
int isis_if_auth_mode_hmac_md5_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name

`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_key_chain_set

This call implements the `isis authentication key-chain` command to configure the key chain to be used for authentication.

Syntax

```
int isis_if_auth_key_chain_set (u_int32_t vr_id, char *name, char *key_chain,
int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`key_chain`	Key chain used for authentication
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_ERR_NO_AUTH_MD5_EXIST` if the authentication mode is not set to MD5 for the level specified.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_if_auth_key_chain_unset

This call implements the `no` form of the `isis authentication key-chain` command to remove the existing key-chain.

Syntax

```
int
isis_if_auth_key_chain_unset (u_int32_t vr_id, char *name, int level)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`key_chain`	Key chain used for authentication
`level`	IS-IS instance level:
`1`	Level-1
`2`	Level-2
`3`	Both Level-1, Level-2

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INVALID_IS_TYPE` when given level is outside of the range.

`ISIS_API_SET_ERR_VR_NOT_EXIST` if the VR does not exist.

`ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED` when any interface parameter is not configured yet.

`ISIS_API_SET_SUCCESS` when the call is successful.

isis_passive_interface_set

This call implements the `passive interface` command to set the interface to passive mode for the current interface.

Syntax

```
int isis_passive_interface_set (u_int32_t vr_id, char *tag, char *name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`name`	Interface name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_LAST_ACTIVE_IF_PASSIVE` if IS-IS cannot make the last active ISIS-IP interface passive.

`ISIS_API_SET_ERR_INVALID_CMD_IF_CLNS_ONLY` if there are no IS-IS enabled interfaces present.

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

isis_passive_interface_unset

This call implements the `no` parameter of the `passive interface` command to reset the interface to active mode for the current interface.

Syntax

```
int isis_passive_interface_unset (u_int32_t vr_id, char *tag, char *name)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.
`name`	Interface name

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_IF_NOT_EXIST` when interface of given name does not exist.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

isis_passive_interface_default_set

This call implements the `passive interface` command to set all interfaces into passive mode, except the high-priority interface.

Syntax

```
int isis_passive_interface_default_set (u_int32_t vr_id, char *tag)
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_LAST_ACTIVE_IF_PASSIVE` if IS-IS cannot make the last active ISIS-IP interface passive.

`ISIS_API_SET_ERR_INVALID_CMD_IF_CLNS_ONLY` if there are no IS-IS enabled interfaces present.

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

isis_passive_interface_default_unset

This call implements the `no` parameter of the `passive interface` command to reset all interfaces to active mode.

Syntax

```
int isis_passive_interface_default_unset (u_int32_t vr_id, char *tag)
```

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
tag	IS-IS instance area tag.

Output Parameters

None

Return Value

`ISIS_API_SET_ERR_INSTANCE_NOT_EXIST` if IS-IS instance does not exist with given tag.

`ISIS_API_SET_SUCCESS` when the call is successful.

`ISIS_API_SET_ERR_VR_NOT_EXIST` when the VR could not be found.

isis_restart_hello_interval_set

This call implements the `isis restart-hello-interval` command to configure the interval of the IS-IS Hello packet with Restart TLV.

Syntax

```
int isis_restart_hello_interval_set (u_int32_t vr_id, char *name, u_int16_t interval,
int level);
```

Input Parameters

Input Parameters

vr_id	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
name	Interface name
interval	Specified interval; default is 3 seconds.
level	IS-IS level

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INVALID_VALUE when the given interval is invalid.

ISIS_API_SET_ERR_INVALID_IS_TYPE when the given IS-IS level is invalid.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_SUCCESS when the call is successful.

isis_restart_hello_interval_unset

This call implements the `no` parameter of the `isis restart-hello-interval` command to reset the interval of the IS-IS Hello packet interval with Restart TLV to the default.

Syntax

```
int isis_restart_hello_interval_unset (u_int32_t vr_id, char *name, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`name`	Interface name
`level`	IS-IS level

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INVALID_IS_TYPE when the given IS-IS level is invalid.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameters are not configured.

ISIS_API_SET_SUCCESS when the call is successful.

isis_restart_level_timer_set

This call implements the `restart-timer` command to configure the maximum timer to wait for the LSP database synchronization.

Syntax

```
int isis_restart_level_timer_set (u_int32_t vr_id, char *tag, u_int16_t timer, int
level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	area tag
`timer`	Expiry timer; the default is 60 seconds.
`level`	IS-IS level

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST when the given area tag does not exist.

ISIS_API_SET_ERR_INVALID_VALUE when the given timer value is invalid.

ISIS_API_SET_ERR_INVALID_IS_TYPE when the given IS-IS level is invalid.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

isis_restart_level_timer_unset

This call implements the `no` parameter of the `restart-timer` command to reset the maximum timer to wait for the LSP database synchronization to the default.

Syntax

```
int isis_restart_level_timer_unset (u_int32_t vr_id, char *tag, int level);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	Area tag
`level`	IS-IS level

Output Parameters

None

Return Value

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST when the given area tag does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE when the given IS-IS level is invalid.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_SUCCESS when the call is successful.

isis_restart_set

This call implements the part of the `restart isis` command to notify NSM to restore the IS-IS routes in the NSM routing table.

Syntax

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

Input Parameter

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`seconds`	Grace period which overrides the current grace period if the value is non-zero; the default is 65535 seconds.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_ERR_GRACE_PERIOD_INVALID when the given grace period is invalid.

isis_restart_grace_period_set

This call implements the `isis restart grace-period` command to configure the grace period.

Syntax

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

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`seconds`	Grace period; the default is 65535 seconds.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_ERR_GRACE_PERIOD_INVALID when the given grace period is invalid.

isis_restart_grace_period_unset

This call implements the `no` parameter of the `isis restart grace-period` command to reset to the default value the grace period.

Syntax

```
int isis_restart_grace_period_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

isis_restart_helper_set

This call implements the `isis restart helper` command to configure the router as the helper router.

Syntax

```
int isis_restart_helper_set (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

isis_restart_helper_unset

This call implements the `no` parameter of the `isis restart helper` command to unconfigure the router as the helper router. This means that a non-helper router initializes adjacency with the restarting router, and recalculates the topology.

Syntax

```
int isis_restart_helper_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS when the call is successful.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

isis_instance_unset_restart

This call implements the part of the `restart isis` command to force shutdown of the IS-IS instance. This stores routes in the NSM, and shuts down the ISIS daemon.

Syntax

```
int isis_instance_unset_restart (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual Router ID. Default value is 0. For non-VR implementation, pass 0 for vr_id.
`tag`	Area tag

Output Parameters

None

Return Value

ISIS_API_SET_ERR_AREA_TAG_TOO_LONG when the given area tag is too long.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST when the given area tag does not exist.

ISIS_API_SET_ERR_VR_NOT_EXIST when the VR does not exist.

ISIS_API_SET_SUCCESS when the call is successful.

CHAPTER 10 IS-IS SNMP API

This chapter includes all of the access able SNMP get and set SNMP API functions.

isis_get_sys_version

This call gets the version number of the IS-IS protocol that this instance implements.

Syntax

```
int
isis_get_sys_max_path_splits (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Pointer to the version strings. One is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_type

This call sets the system type for the instance of the IS-IS protocol.

Syntax

```
int
isis_set_sys_type (u_int32_t vr_id, u_int32_t instance, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Type of IS-IS instance:
1	level1IS
2	level2IS
3	level1L2IS

Output Parameter

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if Virtual Router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the IS-IS instance does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the type of IS-IS instance is invalid.

ISIS_API_SET_ERR_L2_INSTANCE_EXIST if the instance already exists.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_type

This call gets the system type for the instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_type (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Pointer to the version strings. One is returned by default. Values the following:
`1`	Level 1
`2`	Level 2
`3`	Level1 And Level 2

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_get_sys_id

This call gets the system ID for the instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_id (u_int32_t vr_id, u_int32_t instance, u_char **ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Pointer to the system ID string.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_max_path_splits

This call sets the maximum number of paths with equal routing metric values permitted to split between. Only the default value can be set.

Syntax

```
int
isis_set_sys_max_path_splits (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Maximum number of paths with equal routing metric value.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_max_path_splits

This call gets the maximum number of paths with equal routing metric value which it is permitted to split between.

Syntax

```
int
isis_get_sys_max_path_splits (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Maximum number of paths with equal routing metric value. Two is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_max_lsp_gen_interval

This call sets the maximum interval, in seconds, between generated LSPs by this instance of the IS-IS protocol. Only the default value can be set.

Syntax

```
int
isis_set_sys_max_lsp_gen_interval (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Maximum interval between generated LSPs.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_max_lsp_gen_interval

This call gets the maximum interval, in seconds, between generated LSPs by this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_max_lsp_gen_interval (u_int32_t vr_id,
                                    u_int32_t instance, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Maximum interval between generated LSPs. 900 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_max_area_addrs

This call sets the maximum number of area addresses to be permitted for this instance of the IS-IS protocol. Only the default value can be set.

Syntax

```
int
isis_set_sys_max_area_addrs (u_int32_t vr_id, u_int32_t instance,
                             u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Maximum number of area addresses.

Output Parameter

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if Virtual Router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the IS-IS instance does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the given area is not in the range of 3 – 254.

ISIS_API_SET_ERR_TOO_MANY_AREA_ADDRESSES if there are more areas than the maximum.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_max_area_addrs

This call gets the maximum number of area addresses to be permitted for this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_max_area_addrs (u_int32_t vr_id, u_int32_t instance,
                             u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Maximum number of area addresses.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_poll_es_hello_rate

This call sets the value, in seconds, to be used for the suggested ES configuration timer in ISH PDUs when soliciting the ES configuration. Only the default value can be set.

Syntax

```
int
isis_set_sys_poll_es_hello_rate (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Value, in seconds, to be used for the suggested ES configuration timer in ISH PDUs.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_poll_es_hello_rate

This call gets the value, in seconds, to be used for the suggested ES configuration timer in ISH PDUs when soliciting the ES configuration.

Syntax

```
int
isis_get_sys_poll_es_hello_rate (u_int32_t vr_id, u_int32_t instance,
                                 u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Value, in seconds, to be used for the suggested ES configuration timer in ISH PDUs. 50 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_wait_time

This call sets the seconds to delay in waiting state before entering an “on” state. Only the default value can be set.

Syntax

```
int
isis_set_sys_wait_time (u_int32_t vr_id, u_int32_t instance, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Number of seconds to delay in waiting state before “on” state.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_wait_time

This call gets the number of seconds to delay in waiting state before entering the “on” state.

Syntax

```
int
isis_get_sys_wait_time (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Number of seconds to delay in waiting before the "on" state. 60 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_admin_state

This call sets the administrative state of an instance of the IS-IS protocol. Only the default value can be set.

Syntax

```
int
isis_set_sys_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Administrative state:
`1`	On
`2`	Off

Output Parameter

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if Virtual Router does not exist.

ISIS_API_SET_ERR_AREA_TAG_TOO_LONG if the area tag is longer than the maximum length.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the IS-IS instance does not exist.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_admin_state

This call gets the administrative state of this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Administrative state.
`1`	On (default)
`2`	Off

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_log_adj_changes

This call sets the state of the log generation when an IS-IS adjacency changes state (up or down).

Syntax

```
int
```



```
isis_set_sys_log_adj_changes (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	State of the log generation when an IS-IS adjacency changes state:
`1`	True
`2`	False

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_GET_ERROR if set not found.

isis_get_sys_log_adj_changes

This call gets the state of the log generation when an IS-IS adjacency changes state (up or down).

Syntax

```
int isis_get_sys_log_adj_changes (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	State of the log generation when an IS-IS adjacency changes state.
`1`	isisTruthValueTrue
`2`	isisTruthValuefalse

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_get_sys_next_circ_index

This call gets the next ISIS circ index value for this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_log_adj_changes (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Next ISIS circ Index value.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_l2_to_l1_leaking

This call sets the state of the level 2 to level 1 route leaking, for this instance of the IS-IS protocol.

Syntax

```
int
isis_set_sys_l2_to_l1_leaking (u_int32_t vr_id, u_int32_t instance,
                               u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	State of the level 2 to level 1 route leaking.
`1`	True
`2`	False

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_l2_to_l1_leaking

This call gets the state of the level 2 to level 1 route leaking for this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_l2_to_l1_leaking (u_int32_t vr_id, u_int32_t instance,
                               u_int32_t *ret);
```

Input Parameters

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

`instance`	IS-IS instance ID.

Output Parameters

`ret`	state of the level 2 to level 1 route leaking:
`1`	True
`2`	False (default)

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_max_age

This call sets the value for the RemainingLifeTime field of the LSP, which is generated by an instance of IS-IS.

Syntax

```
int
isis_set_sys_max_age (u_int32_t vr_id, u_int32_t instance, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Value to place in RemainingLifeTime field of an LSP.

Output Parameter

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if Virtual Router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the IS-IS instance does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the given area is not in the range of 1 – 65535.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_max_age

This call gets the system max age value for LSPs generated by this instance of the IS-IS protocol.

Syntax

```
int
isis_get_sys_max_age (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Returns the “RemainingLifeTime” value of an LSP.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_receive_lsp_bufsize

This call sets the size of the largest buffer this instance can store.

Syntax

```
int
isis_set_sys_receive_lsp_bufsize (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	Size of the largest receive buffer. only ISIS_PDU_MAX_LENGTH can be set.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_receive_lsp_bufsize

This call gets the size of the largest buffer this instance can store.

Syntax

```
int
isis_get_sys_receive_lsp_bufsize (u_int32_t vr_id,
                                  u_int32_t instance, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	Size of the largest receive buffer. ISIS_PDU_MAX_LENGTH is returned by default.

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_set_sys_exist_state

This call sets the state of the IS-IS router of this instance.

Syntax

```
int isis_set_sys_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`val`	State of the IS-IS router.

Output Parameter

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if Virtual Router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the IS-IS instance does not exist.

ISIS_API_SET_ERR_AREA_TAG_TOO_LONG if the tag length is longer than the maximum length.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; instance not found.

isis_get_sys_exist_state

This call gets the state of the IS-IS router of this instance.

Syntax

```
int
isis_get_sys_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

Output Parameters

`ret`	State of the IS-IS router:
1	Active
2	Not in service
3	Not ready
4	Create and go
5	Create and wait
6	Destroy

Return Values

ISIS_API_GET_SUCCESS for instance found.

ISIS_API_GET_ERROR for instance not found.

isis_get_next_sys_version

This call gets the version number of the IS-IS protocol that the next instance implements.

Syntax

```
int
isis_get_next_sys_version (u_int32_t vr_id, u_int32_t *instance,
                           int indexlen, char **ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the version strings. “1” is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_type

This call gets the system type for the next instance of the IS-IS protocol.

Syntax

```
int isis_get_next_sys_type (u_int32_t vr_id, u_int32_t *instance, int indexlen,
u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
indexlen	Length of the index.

Output Parameters

ret	Type of IS-IS instance:
1	Level 1
2	Level 2
3	Level1 and level 2

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_id

This call gets the system ID for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_type (u_int32_t vr_id, u_int32_t *instance,
                        int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
indexlen	Length of the index.

Output Parameters

ret	Pointer to the system ID string.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_max_path_splits

This call gets the maximum number of paths with an equal routing metric value, which is permitted to split between for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_max_path_splits (u_int32_t vr_id, u_int32_t *instance,
                                   int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
indexlen	Length of the index.

Output Parameters

ret	Maximum number of paths with equal routing metric value. “2” is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_max_lsp_gen_interval

This call gets the maximum interval, in seconds, between generated LSPs by the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_max_lsp_gen_interval (u_int32_t vr_id, u_int32_t *instance,
                                        int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Maximum interval between generated LSPs. “900” is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_max_area_addrs

This call gets the maximum number of area addresses to be permitted for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_max_area_addrs (u_int32_t vr_id, u_int32_t *instance,
                                  int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Maximum number of area addresses.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_poll_es_hello_rate

This call gets the value, in seconds, for the suggested ES configuration timer in ISH PDUs when soliciting the ES configuration for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_poll_es_hello_rate (u_int32_t vr_id, u_int32_t *instance,
                                      int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Value, in seconds, to be used for the suggested ES configuration timer in ISH PDUs. 50 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_wait_time

This call gets the number of seconds to delay in waiting before entering a state for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_wait_time (u_int32_t vr_id, u_int32_t *instance,
                             int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of seconds to delay in waiting state before “on” state. 60 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_admin_state

This call gets the administrative state of the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_admin_state (u_int32_t vr_id, u_int32_t *instance,
                               int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Administrative state:
`1`	On (default)
`2`	Off

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_log_adj_changes

This call gets the state of the log generation when an IS-IS adjacency changes state (up or down) for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_log_adj_changes (u_int32_t vr_id, u_int32_t *instance,
                                   int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the log generation when an IS-IS adjacency changes state:
`1`	True
`2`	False

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_next_circ_index

This call gets the next isisCircIndex value for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_next_circ_index (u_int32_t vr_id, u_int32_t *instance,
                                   int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Next isisCircIndex value.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_l2_to_l1_leaking

This call gets the state of the level 2 to level 1 route leaking for the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_l2_to_l1_leaking (u_int32_t vr_id, u_int32_t *instance,
                                    int indexlen, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the level 2 to level 1 route leaking:
`1`	True
`2`	False (default)

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_max_age

This call gets the value to place of LSPs, which is generated by the next instance of the IS-IS protocol.

Syntax

```
int
isis_get_next_sys_max_age (u_int32_t vr_id, u_int32_t *instance,
                           int indexlen, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Value to place in RemainingLifeTime field of LSPs.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_receive_lsp_bufsize

This call gets the size of the largest buffer the next instance can store.

Syntax

```
int
isis_get_next_sys_receive_lsp_bufsize (u_int32_t vr_id, u_int32_t *instance,
                                       int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	Size of the largest receive buffer. ISIS_PDU_MAX_LENGTH is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_get_next_sys_exist_state

This call gets the state of the IS-IS router of the next instance.

Syntax

```
int isis_get_next_sys_exist_state (u_int32_t vr_id, u_int32_t *instance, int indexlen, 
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the IS-IS router:
1	Active (default)
2	Not in service
3	Not ready
4	Create and go
5	Create and wait
6	Destroy

Return Values

ISIS_API_GET_SUCCESS for the next instance found.

ISIS_API_GET_ERROR for the next instance not found.

isis_set_man_area_addr_state

This call sets the state of the manually configured area address.

Syntax

```
int isis_set_man_area_addr_state (u_int32_t vr_id, u_int32_t instance, struct 
isis_area_addr addr, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`addr`	A variable length of a manually configured area address.
`val`	State of the manually configured area address:
1	Active
2	NotInService
3	NotReady
4	CreateAndGo
5	CreateAndWait
6	Destroy

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete in the specified instance.

ISIS_API_SET_ERROR for set not complete in the specified instance; area address not found.

isis_get_man_area_addr_state

This call gets the state of the manually configured area address.

Syntax

```
int isis_get_man_area_addr_state (u_int32_t vr_id, u_int32_t instance, struct
isis_area_addr addr, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the manually configured area address:
1	Active (default)
2	Not in service
3	Not ready
4	Create and go
5	Create and wait
6	Destroy

Return Values

ISIS_API_GET_SUCCESS for area address found in the specified instance.

ISIS_API_GET_ERROR for area address not found in the specified instance.

isis_get_next_man_area_addr_state

This call gets the state of the next manually configured area address.

Syntax

```
int isis_get_next_man_area_addr_state (u_int32_t vr_id, u_int32_t *instance, struct
isis_area_addr *addr, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the manually configured area address:

1	Active (default)
2	Not in service
3	Not ready
4	Create and go
5	Create and wait
6	Destroy

Return Values

ISIS_API_GET_SUCCESS for the next area address found.

ISIS_API_GET_ERROR for the next area address not found.

isis_get_sys_area_addr

This call gets the area address reported in a level 1 LSP received by this instance of the IS-IS protocol.

Syntax

```
int isis_get_sys_area_addr (u_int32_t vr_id, u_int32_t instance, struct isis_area_addr
addr, struct isis_area_addr **ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
indexlen	Length of the index.

Output Parameters

ret	Area address reported in a level 1 LSP received by this instance of the IS-IS protocol.

Return Values

ISIS_API_GET_SUCCESS for area address found in the specified instance.

ISIS_API_GET_ERROR for area address not found in the specified instance.

isis_get_next_sys_area_addr

This call gets the next area address reported in a level 1 LSP received by the instance of the IS-IS protocol.

Syntax

```
int isis_get_next_sys_area_addr (u_int32_t vr_id, u_int32_t *instance, struct
isis_area_addr *addr, int indexlen, struct isis_area_addr **ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
addr	Variable length area address expressed in the isis_area_addr structure.
indexlen	Length of the index.

Output Parameters

`ret`	Area address reported in a level 1 LSP received by this instance of the IS-IS protocol.

Return Values

ISIS_API_GET_SUCCESS for the next area address found.

ISIS_API_GET_ERROR for the next area address not found.

isis_set_prot_supp_exist_state

This call gets the state of the supported protocol.

Syntax

```
int isis_set_prot_supp_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
protocol, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`protocol`	Supported protocol:
`129`	ISO8473
`204`	IP
`142`	IPv6
`val`	State of the manually configured supported protocol:
`1`	Active
`2`	NotInService
`3`	NotReady
`4`	CreateAndGo
`5`	CreateAndWait
`6`	Destroy

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete in the specified instance.

ISIS_API_SET_ERROR for set not complete in the specified instance.

isis_get_prot_supp_exist_state

This call gets the state of the supported protocol.

Syntax

```
int isis_get_prot_supp_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
protocol, u_int32_t *ret);
```


Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`protocol`	Supported protocol:
`129`	ISO8473
`204`	IP
`142`	IPv6

Output Parameters

`ret`	State of the manually configured supported protocol:
`1`	Active (default)
`2`	Not in service
`3`	Not ready
`4`	Create and go
`5`	Create and wait
`6`	Destroy

Return Values

ISIS_API_GET_SUCCESS for supported protocol found in the specified instance.

ISIS_API_GET_ERROR for supported protocol not found in the specified instance.

isis_get_next_prot_supp_exist_state

This call gets the state of the next supported protocol.

Syntax

```
int isis_get_next_prot_supp_exist_state (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *protocol, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`protocol`	Supported protocol:
`129`	ISO8473
`204`	IP
`142`	IPv6
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the manually configured supported protocol:
`1`	Active (default)
`2`	Not in service

3	Not ready
4	Create and go
5	Create and wait
6	Destroy

Output Parameters

Return Values

ISIS_API_GET_SUCCESS for next supported protocol found in the instance.

ISIS_API_GET_ERROR for next supported protocol not found in the instance.

isis_set_summ_addr_state

This call sets the existence state of this summary address.

Syntax

```
int
isis_set_summ_addr_state (u_int32_t vr_id, u_int32_t instance, u_int32_t type,
                          struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
type	Type of summary IP address.
p	Summary IP address.
prefixlen	Prefix length of summary IP address.
val	Existence state of this summary address:
	isisRowStatusActive
	isisRowStatusDestroy
	isisRowStatusCreateAndGo

Output Parameter

None

Return value

ISIS_API_SET_SUCCESS for the state of summary IP address set in the specified instance;

ISIS_API_SET_ERROR for the state of the summary IP address not set in the specified instance.

isis_get_summ_addr_state

This call gets the existence state of this summary address.

Syntax

```
int isis_get_summ_addr_state (u_int32_t vr_id, u_int32_t instance, u_int32_t type,
struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```


Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
type	Type of summary IP address.
p	Summary IP address.
prefixlen	Prefix length of summary IP address.

Output Parameter

ret	Existence state of this summary address. Active is returned by default.

Return value

ISIS_API_GET_SUCCESS for summary IP address found in the specified instance.

ISIS_API_GET_ERROR for summary IP address not found in the specified instance.

isis_set_summ_addr_metric

This call sets the metric value to announce this summary address.

Syntax

```
int isis_set_summ_addr_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t type,
struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
type	Type of summary IP address.
p	Summary IP address.
prefixlen	Prefix length of summary IP address.
val	Wide metric value to announce this summary address.

Output Parameter

None

Return value

ISIS_API_SET_SUCCESS for the metric of summary IP address set in the specified instance.

ISIS_API_SET_ERROR for the metric of summary IP address not set in the specified instance.

isis_get_summ_addr_metric

This call gets the metric value to announce this summary address.

Syntax

```
int isis_get_summ_addr_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t type,
struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.

Output Parameter

`ret`	Metric value to announce this summary address.

Return value

ISIS_API_GET_SUCCESS for summary IP address found in the specified instance.

ISIS_API_GET_ERROR for summary IP address not found in the specified instance.

isis_set_summ_addr_full_metric

This call sets the wide metric value to announce this summary address.

Syntax

```
int isis_set_summ_addr_full_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t
type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.
`val`	Wide metric value to announce this summary address.

Output Parameter

None

Return value

ISIS_API_SET_SUCCESS for the full metric of summary IP address set in the specified instance

ISIS_API_SET_ERROR for the full metric of summary IP address not set in the specified instance.

isis_get_summ_addr_full_metric

This call gets the wide metric value to announce this summary address.

Syntax

```
int isis_get_summ_addr_full_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t
type, struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```


Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.

Output Parameter

`ret`	Wide metric value to announce this summary address.

Return value

ISIS_API_GET_SUCCESS for summary IP address found in the specified instance.

ISIS_API_GET_ERROR for summary IP address not found in the specified instance.

isis_get_next_summ_addr_state

This call gets the next the existence state of this summary address.

Syntax

```
int isis_get_next_summ_addr_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*type, struct prefix *p, u_int32_t *prefixlen, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.
`indexlen`	Length of the index.

Output Parameter

`ret`	Existence state of this summary address. Active is returned by default.

Return value

ISIS_API_GET_SUCCESS for the summary IP address found in the specified instance.

ISIS_API_GET_ERROR for the next summary IP address not found in the specified instance.

isis_get_next_summ_addr_metric

This call gets the metric value to announce this summary address.

Syntax

```
int isis_get_next_summ_addr_metric (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*type, struct prefix *p, u_int32_t *prefixlen, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.
`indexlen`	Length of the index.

Output Parameter

`ret`	Metric value to announce this summary address.

Return value

ISIS_API_GET_SUCCESS for the summary IP address found in the specified instance.

ISIS_API_GET_ERROR for the next summary IP address not found in the specified instance.

isis_get_next_summ_addr_full_metric

This call gets the next, wide metric value to announce this summary address.

Syntax

```
int isis_get_next_summ_addr_full_metric (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`type`	Type of summary IP address.
`p`	Summary IP address.
`prefixlen`	Prefix length of summary IP address.
`indexlen`	Length of the index.

Output Parameter

`ret`	Wide metric value to announce this summary address.

Return value

ISIS_API_GET_SUCCESS for the summary IP address found in the specified instance.

ISIS_API_GET_ERROR for the next summary IP address not found in the specified instance.

isis_set_sys_level_lsp_bufsize

This call sets the maximum size of LSPs and SNPs originated by the instance of the IS-IS protocol at this level.

Syntax

```
int
```



```
isis_set_sys_level_lsp_bufsize (u_int32_t vr_id, u_int32_t instance,
                                u_int32_t level,
                                u_int32_t val)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	Maximum size of LSPs and SNPs.

Output Parameter

None

Return Values

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found, or given `bufsize` is not the range of isisLSPBuffSizeMin – isisLSPBuffSizeMax.

isis_get_sys_level_lsp_bufsize

This call gets maximum size of LSPs and SNPs originated by the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_lsp_bufsize (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	Maximum size of LSPs and SNPs.

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_min_lsp_gen_interval

This call sets the minimum interval, in seconds, between successive generation of LSPs with the same LSP ID by the instance of the IS-IS protocol at this level.

Syntax

```
int isis_set_sys_level_min_lsp_gen_interval (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t val);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	Minimum interval, in seconds, between successive generation of LSPs. Range is 1 – 120.

Output Parameter

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the given value is not in the range of 1 – 120.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the IS – LEVEL is invalid.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_min_lsp_gen_interval

This call gets the minimum interval, in seconds, between successive generation of LSPs with the same LSP ID by the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_min_level_lsp_gen_interval (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	Minimum interval, in seconds, between successive generation of LSPs.

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_get_sys_level_overload_state

This call gets the state of the database for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_overload_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```


Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	State of the level 1 database:
`1`	isisLevelStateOff
`2`	isisLevelStateOn
`3`	isisLevelStateWaiting
`4`	isisLevelStateOverloaded

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_set_overload

This call sets the state of the overload bit for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_set_sys_level_set_overload (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t val);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	State of the overload bit:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_set_overload

This call gets the state of the overload bit for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_set_overload (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	State of the overload bit:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_set_overload_until

This call sets the time, in seconds, the overload bit should be set for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_set_sys_level_set_overload_until (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t val);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	Time, in seconds, the overload bit should be set.

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_ERR_OVERLOAD_INTERVAL_INVALID if the given interval is not in the range of 5 – 86400.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_set_overload_until

This call gets the time, in seconds, the overload bit should be set for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_set_overload_until (u_int32_t vr_id, u_int32_t instance, 
u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	Time, in seconds, the overload bit should be set.

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_metric_style

This call sets the metric style for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_set_sys_level_metric_style (u_int32_t vr_id, u_int32_t instance, u_int32_t 
level, u_int32_t val);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	Metric style at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_metric_style

This call gets the metric style for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_metric_style (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	Metric style at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_spf_considers

This call sets the type of metric to consider in the SPF computation for an IS-IS instance at this level.

Syntax

```
int
isis_set_sys_level_spf_considers (u_int32_t vr_id,
                                  u_int32_t instance, u_int32_t level,
                                  u_int32_t val)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	Metric type to use in the SPF computation at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_spf_considers

This call gets the metric to be considered in the SPF computation for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_spf_considers (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

Output Parameters

`ret`	Metric to use in the SPF computation at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_set_sys_level_te_enabled

This call sets the state of the traffic engineering for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_set_sys_level_te_enabled (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t val);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`val`	State of the traffic engineering at this level:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Output Parameters

None

Return Values

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INSTANCE_NOT_EXIST if the instance does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the IS-LEVEL is not valid.

ISIS_API_SET_ERR_TE_ENABLED if Traffic Engineering is enabled.

ISIS_API_SET_ERR_WIDE_METRIC_NOT_SET if the wide metric is not set.

ISIS_API_SET_SUCCESS for the specified level found.

ISIS_API_SET_ERROR for the level not found.

isis_get_sys_level_te_enabled

This call gets the state of the traffic engineering for the instance of the IS-IS protocol at this level.

Syntax

```
int isis_get_sys_level_te_enabled (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index.

Output Parameters

ret	State of the traffic engineering at this level:
1	isisTruthValueTrue
2	isisTruthValueFalse

Return Values

ISIS_API_GET_SUCCESS for the specified level found.

ISIS_API_GET_ERROR for the level not found.

isis_get_next_sys_level_lsp_bufsize

This call gets the next maximum size of LSPs and SNPs originated by an instance of IS-IS at the next level.

Syntax

```
int isis_get_next_sys_level_lsp_bufsize (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.

`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Maximum size of LSPs and SNPs.

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_min_lsp_gen_interval

This call gets the next minimum interval, in seconds, between successive generation of LSPs with the same LSP ID by the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_min_lsp_gen_interval (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Minimum interval, in seconds, between successive generation of LSPs.

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_overload_state

This call gets the next the state of the database for the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_overload_state (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the level 1 database:
`1`	isisLevelStateOff
`2`	isisLevelStateOn
`3`	isisLevelStateWaiting
`4`	isisLevelStateOverloaded

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_set_overload

This call gets the next state of the overload bit for the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_set_overload (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the overload bit:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_set_overload_until

This call gets the time, in seconds, the overload bit should be set for the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_set_overload_until (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`level`	IS-IS level index.

Output Parameters

`ret`	Time, in seconds, the overload bit should be set.

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_metric_style

This call gets the next metric style for the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_metric_style (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Metric style at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_spf_considers

This call gets the type of metric to be considered in an SPF computation for the instance of IS-IS at the next level.

Syntax

```
int isis_get_next_sys_level_spf_considers (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.

`indexlen`	Length of the index.

Output Parameters

`ret`	Metric to be considered in the SPF computation at this level:
`1`	isisMetricStyleNarrow
`2`	isisMetricStyleWide
`3`	isisMetricStyleBoth

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_get_next_sys_level_te_enabled

This call gets the next state of the traffic engineering for the instance of the IS-IS protocol at the next level.

Syntax

```
int isis_get_next_sys_level_te_enabled (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the traffic engineering at this level:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Return Values

ISIS_API_GET_SUCCESS for the specified next level found.

ISIS_API_GET_ERROR for the next level not found.

isis_set_circ_ifindex

This call sets the value of interface index for an interface for a corresponding circuit. The interface index cannot be changed.

Syntax

```
int isis_set_circ_ifindex (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameter

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	Interface index that corresponds to the circuit index.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_ifindex

This call gets the value of interface index for the interface to which this circuit corresponds.

Syntax

```
int isis_get_circ_ifindex (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	Interface index that corresponds to the circuit index.

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_admin_state

This call sets the administrative state of the circuit.

Syntax

```
int isis_set_circ_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	Administrative state:
`1`	isisAdminStateOn (default)

2	isisAdminStateOff

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_admin_state

This call gets the administrative state of the circuit.

Syntax

```
int isis_get_circ_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.

Output Parameters

ret	Administrative state:
1	isisAdminStateOn (default)
2	isisAdminStateOff

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_exist_state

This call sets the existence state of the circuit.

Syntax

```
int isis_set_circ_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t val);
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
val	State of the specified circuit.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_exist_state

This call gets the existence state of the circuit.

Syntax

```
int isis_get_circ_exist_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.

Output Parameters

ret	State of the specified circuit:
1	isisRowStatusActive (default)
2	isisRowStatusNotInservice
3	isisRowStatusNotReady
4	isisRowStatusCreateAndGo
5	isisRowStatusCreateAndWait
6	isisRowStatusDestroy

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_type

This call sets the type of the circuit. only broadcast and point-to-point type circuits are supported.

Syntax

```
int isis_set_circ_type (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.

`val`	Type of the specified circuit.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_type

This call gets the type of a circuit.

Syntax

```
int isis_get_circ_type (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	Type of the specified circuit.

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_ext_domain

This call sets the status of the normal transmission of and interpretation of intra-domain IS-IS PDUs on this circuit.

Syntax

```
int isis_set_circ_ext_domain (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	State of the intra-domain IS-IS PDUs:
`1`	isisTruthValueFalse (default)
`2`	isisTruthValueTrue

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_ext_domain

This call gets the status of the normal transmission and interpretation of intra-domain IS-IS PDUs on this circuit.

Syntax

```
int isis_get_circ_ext_domain (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	State of the intra-domain IS-IS PDUs:
1	isisTruthValueFalse (default)
2	isisTruthValueTrue

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_level

This call sets the type of packets that will be sent and accepted on this circuit.

Syntax

```
int isis_set_circ_level (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	Level of the circuit:
1	Level1
2	Level2
3	Level1 and Level 2

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist

ISIS_API_SET_ERR_INVALID_IS_TYPE if the circuit level is invalid

ISIS_API_SET_ERR_IF_NOT_EXIST if the interface does not exist

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_level

This call gets the type of packets that will be sent and accepted on this circuit.

Syntax

```
int isis_get_circ_level (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	An IS-IS circuit index.

Output Parameters

`ret`	Level of the circuit:
1	Level1
2	Level2
3	Level1 and Level 2

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_passive_if

This call gets the status to include this circuit in LSPs, even if it is not running the IS-IS protocol.

Syntax

```
int isis_set_circ_passive_if (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameter

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`ret`	To include this circuit in LSPs, even if it is not running the IS-IS protocol:
`1`	isisTruthValueFalse (default)
`2`	isisTruthValueTrue

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_IF_NOT_EXIST interface does not exist.

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_passive_if

This call gets the status to include this circuit in LSPs, even if it is not running the IS-IS protocol.

Syntax

```
int isis_get_circ_passive_if (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	To include this circuit in LSPs, even if it is not running the IS-IS protocol:
`1`	isisTruthValueFalse (default)
`2`	isisTruthValueTrue

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_mesh_enabled

This call gets the status of the mesh group configuration of this circuit.

Syntax

```
int
```

```
isis_set_circ_mesh_enabled (u_int32_t vr_id, u_int32_t instance,
                            u_int32_t circindex,
                            u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	To include this circuit in LSPs, even if it is not running the IS-IS protocol:
`1`	isisMeshGroupInactive
`2`	isisMeshGroupBlocked
`3`	isisMeshGroupSet

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_IF_PARAM_NOT_CONFIGURED if the interface parameters are not configured.

ISIS_API_SET_ERR_IF_NOT_EXIST interface does not exist.

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_mesh_enabled

This call gets the status of the mesh group configuration of this circuit.

Syntax

```
int isis_get_circ_mesh_enabled (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	To include this circuit in LSPs, even if it is not running the IS-IS protocol:
`1`	isisMeshGroupInactive
`2`	isisMeshGroupBlocked
`3`	isisMeshGroupSet

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_mesh_group

This call gets the identifier of the mesh group of this circuit.

Syntax

```
int isis_set_circ_mesh_group (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	Mesh group ID.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_MESH_GROUP_ID_INVALID if the mesh group ID is invalid.

ISIS_API_SET_ERR_IF_NOT_EXIST interface does not exist.

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_mesh_group

This call gets the identifier of the mesh group of this circuit.

Syntax

```
int
isis_get_circ_mesh_group (u_int32_t vr_id,
                          u_int32_t instance, u_int32_t circindex,
                          u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	Mesh group ID.

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_small_hellos

This call gets the status of the IS-IS LAN hellos padding of this circuit.

Syntax

```
int isis_set_circ_small_hellos (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`val`	Value indicates whether unpadded hellos can be sent on LAN circuits.
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse (default)

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_small_hellos

This call gets the status of the IS-IS LAN hellos padding of this circuit.

Syntax

```
int isis_get_circ_small_hellos (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	Value indicates whether unpadded hellos can be sent on LAN circuits.

1	isisTruthValueTrue
2	isisTruthValueFalse (default)

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_get_circ_uptime

This call gets the amount of time, in seconds, since this circuit entered state 'up' if the circuit is up, or the number of seconds since the circuit was up if the circuit is not up, or since the system started if the circuit has never been up.

Syntax

```
int isis_get_circ_uptime (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex, 
u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.

Output Parameters

ret	Seconds since the object has been 'up'. If the object is not up, seconds since the circuit was up or since the system started if the circuit has never been up.

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_set_circ_3way_enabled

This call gets the status of this circuit enabled 3Way handshake.

Syntax

```
int isis_set_circ_3way_enabled (u_int32_t vr_id, u_int32_t instance, u_int32_t 
circindex, u_int32_t val);
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
val	Status of the circuit enabled 3Way handshake.
1	isisTruthValueTrue
2	isisTruthValueFalse (default)

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., supported circuit index not found in the specified instance.

isis_get_circ_3way_enabled

This call gets the status of this circuit enabled 3Way handshake.

Syntax

```
int isis_get_circ_3way_enabled (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

Output Parameters

`ret`	Status of the circuit enabled 3Way handshake.
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse (default)

Return Values

ISIS_API_GET_SUCCESS for supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for supported circuit index not found in the specified instance.

isis_get_next_circ_ifindex

This call gets the value of interface index for the interface to which the next circuit corresponds.

Syntax

```
int isis_get_next_circ_ifindex (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Interface index that corresponds to the next circuit index.

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_ifsubindex

This call gets a specifier for the part of an interface index that the next circuit corresponds, such as DLCI or VPI/VCI.

Syntax

```
int
isis_get_next_circ_ifsubindex (u_int32_t vr_id, u_int32_t *instance,
                               u_int32_t *circindex,
                               int indexlen, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Interface sub-index that corresponds to the next circuit index.

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_admin_state

This call gets the administrative state of the next circuit.

Syntax

```
int isis_get_next_circ_admin_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Administrative state of the next circuit:
`1`	isisAdminStateOn (default)
`2`	isisAdminStateOff

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_exist_state

This call gets the existence state of the next circuit.

Syntax

```
int isis_get_next_circ_exist_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the next circuit.
1	isisRowStatusActive (default)
2	isisRowStatusNotInservice
3	isisRowStatusNotReady
4	isisRowStatusCreateAndGo
5	isisRowStatusCreateAndWait
6	isisRowStatusDestroy

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_type

This call gets the type of the next circuit.

Syntax

```
int
isis_get_next_circ_type (u_int32_t vr_id, u_int32_t *instance,
                         u_int32_t *circindex,
                         int indexlen, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Type of the next circuit.

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_ext_domain

This call gets the status of the normal transmission of and interpretation of intra-domain IS-IS PDUs on the next circuit.

Syntax

```
int isis_get_next_circ_ext_domain (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the intra-domain IS-IS PDUs of the next circuit:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse (default).

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_level

This call gets the type of packets that will be sent and accepted on the next circuit.

Syntax

```
int isis_get_next_circ_level (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

`indexlen`	Length of the index.

Output Parameters

`ret`	Level of the next circuit.
`1`	Level1
`2`	Level2
`3`	Level1 and level 2

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_passive_if

This call gets the status to include the next circuit in LSPs, even if it is not running the IS-IS protocol.

Syntax

```
int isis_get_next_circ_passive_if (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Include the circuit in LSPs, even if it is not running the IS-IS protocol:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse (default)

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_mesh_enabled

This call gets the status of the mesh group configuration of the next circuit.

Syntax

```
int isis_get_next_circ_mesh_enabled (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of the mesh group for the next circuit.
`1`	isisMeshGroupInactive
`2`	isisMeshGroupBlocked
`3`	isisMeshGroupSet

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_mesh_group

This call gets the identifier of the mesh group of the next circuit.

Syntax

```
int isis_get_next_circ_mesh_group (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Mesh group ID.

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_small_hellos

This call gets the status of the IS-IS LAN hellos padding of the next circuit.

Syntax

```
int isis_get_next_circ_small_hellos (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	State of sending updated hello packets on the next circuit:
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_uptime

This call gets the amount of time in seconds since the circuit entered state 'up' if the circuit is up, or the number of seconds since the circuit was up if the circuit is not up, or since the system started if the circuit has never been up.

Syntax

```
int isis_get_next_circ_uptime (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Seconds since the object has entered the state 'up'. If the object is not up, seconds since the circuit was up, or since the system started, if the circuit has never been up.

Return Values

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_get_next_circ_3way_enabled

This call gets the status of the next circuit enabled 3Way handshake.

Syntax

```
int isis_get_next_circ_3way_enabled (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, int indexlen, u_int32_t *ret);
```

Input Parameter

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Status of the circuit enabled 3Way handshake.
`1`	isisTruthValueTrue
`2`	isisTruthValueFalse (default)

Return Value

ISIS_API_GET_SUCCESS for the next supported circuit index found in the specified instance.

ISIS_API_GET_ERROR for the next supported circuit index not found in the specified instance.

isis_set_circ_level_metric

This call sets the metric value of this circuit for this level.

Syntax

```
int isis_set_circ_level_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`ret`	Sub-range for default metric for single hop which picks between 0 to 63.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_metric

This call gets the metric value of this circuit for this level.

Syntax

```
int
isis_get_circ_level_metric (u_int32_t vr_id, u_int32_t instance,
                            u_int32_t circindex, u_int32_t level,
                            u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Sub-range for default metric for single hop which picks between 0 to 63.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_wide_metric

This call sets the wide metric value of this circuit for this level.

Syntax

```
int isis_set_circ_level_wide_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	Wide metric for IS neighbors which pick between 0 to 1,677,215.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the metric is not in the range of 0 to 1677215.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_wide_metric

This call gets the wide metric value of this circuit for this level.

Syntax

```
int isis_get_circ_level_wide_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Wide metric for IS neighbors which pick between 0 to 1,677,215.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_priority

This call sets the priority for becoming LAN designated IS at this level.

Syntax

```
int isis_set_circ_level_priority (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	priority for becoming LAN designated IS.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_ERR_INVALID_VALUE if the value is not in the range of 0 to 127

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_priority

This call gets the priority for becoming the LAN designated IS at this level.

Syntax

```
int isis_get_circ_level_priority (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Sub-range for IS-IS priority.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_id_octet

This call sets a one-byte identifier that is used in protocol packets to identify a circuit for this level. The level ID octet cannot be changed.

Syntax

```
int isis_set_circ_level_id_octet (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	One-byte identifier that is used in protocol packets to identify a circuit.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_id_octet

This call gets a one-byte identifier that can be used in protocol packets to identify a circuit for this level.

Syntax

```
int isis_get_circ_level_id_octet (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	One-byte identifier. It can be used in protocol packets to identify a circuit.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_get_circ_level_id

This call gets the ID of the circuit allocated during initialization.

Syntax

```
int isis_get_circ_level_id (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t level, struct isis_dis_id *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Pointer to the ID for a circuit.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_get_circ_level_dis

This call gets the ID of the LAN designated IS on this circuit at this level.

Syntax

```
int isis_get_circ_level_dis (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t level, struct isis_dis_id **ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Pointer to the LAN designated IS ID.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_hello_multiplier

This call sets the hello multiplier which is multiplied by the corresponding HelloTimer, and the result, in seconds (rounded up), is used as the holding time in transmitted hellos, to be used by receivers of hello packets from this IS.

Syntax

```
int isis_set_circ_level_hello_multiplier (u_int32_t vr_id, u_int32_t instance,
u_int32_t circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	Hello multiplier.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_ERR_INVALID_VALUE if the value is not in the range of 2 to 100.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_hello_multiplier

This call gets the hello multiplier that is multiplied by the corresponding HelloTimer; and the result in seconds (rounded up) is used as the holding time in transmitted hellos, to be used by receivers of hello packets from this IS.

Syntax

```
int isis_get_circ_level_hello_multiplier (u_int32_t vr_id, u_int32_t instance,
u_int32_t circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Hello multiplier.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index;

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_hello_timer

This call sets the maximum period, in milliseconds, between IIH PDUs on multiaccess networks at this level for LANs. The value at level 1 is used as the period between Hellos on L1L2 point-to-point circuits.

Syntax

```
int isis_set_circ_level_hello_timer (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	Maximum period, in milliseconds, between IIH PDUs on multiaccess networks at this level for LANs. The minimum value is 1000 or 1 second. The value at level 1 is used as the period between hellos on L1L2 point to point circuits.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_ERR_INVALID_VALUE if the value is not in the range of 1 to 65535.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_hello_timer

This call gets the maximum period, in milliseconds, between IIH PDUs on multi-access networks at this level for LANs. The value at level 1 is used as the period between Hellos on L1L2 point-to-point circuits.

Syntax

```
int isis_get_circ_level_hello_timer (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Period, in milliseconds, between IIH PDUs on multiaccess networks at this level for LANs. The value at level 1 is used as the period between Hellos on L1L2 point to point circuits.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index;

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_dis_hello_timer

This call sets the period, in milliseconds, between hello PDUs on multiaccess networks when this is the designated IS.

Syntax

```
int
isis_set_circ_level_dis_hello_timer (u_int32_t vr_id,
                                      u_int32_t instance, u_int32_t circindex,
                                      u_int32_t level, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	hello timer of designated IS.

Output Parameters

none

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_ERR_INVALID_VALUE if the value is not in the range of 1 to 65535.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_dis_hello_timer

This call gets the period, in milliseconds, between hello PDUs on multiaccess networks when this is the designated IS.

Syntax

```
int isis_get_circ_level_dis_hello_timer (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	Hello timer of designated IS.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_lsp_throttle

This call sets minimal interval of time, in milliseconds, between transmissions of LSPs on an interface at this level.

Syntax

```
int
isis_set_circ_level_lsp_throttle (u_int32_t vr_id,
                                   u_int32_t instance, u_int32_t circindex,
                                   u_int32_t level, u_int32_t val)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
val	LSP minimum interval.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the value is greater than 65535.

ISIS_API_SET_ERR_IF_NOT_EXIST if the interface does not exist.

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_lsp_throttle

This call gets the minimal interval of time, in milliseconds, between transmissions of LSPs on an interface at this level.

Syntax

```
int
isis_get_circ_level_lsp_throttle (u_int32_t vr_id, u_int32_t instance,
                                   u_int32_t circindex, u_int32_t level,
                                   u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	LSP minimum interval.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_min_lsp_retrans

This call sets minimum interval, in seconds, between re-transmission of an LSP at this level.

Syntax

```
int isis_set_circ_level_min_lsp_retrans (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	Minimum LSP retransmission interval.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the value is greater than 65535.

ISIS_API_SET_ERR_IF_NOT_EXIST if the interface does not exist.

ISIS_API_SET_ERR_IF_NOT_ENABLED if the interface is not enabled.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_min_lsp_retrans

This call gets the minimum interval, in seconds, between re-transmission of an LSP at this level.

Syntax

```
int isis_get_circ_level_min_lsp_retrans (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Minimum LSP re-transmission interval.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_csnp_interval

This call sets the interval of time, in seconds, between transmission of CSNPs on multiaccess networks if this router is the designated IS at this level.

Syntax

```
int isis_set_circ_level_csnp_interval (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`val`	CSNP interval.

Output Parameters

None

Return Value

ISIS_API_SET_ERR_VR_NOT_EXIST if the virtual router does not exist.

ISIS_API_SET_ERR_INVALID_VALUE if the value is greater than 65535.

ISIS_API_SET_ERR_INVALID_IS_TYPE if the LEVEL is invalid.

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_csnp_interval

This call gets the interval of time, in seconds, between transmission of CSNPs on multiaccess networks if this router is the designated IS at this level.

Syntax

```
int isis_get_circ_level_csnp_interval (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	CSNP interval.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_set_circ_level_psnp_interval

This call gets the minimum interval in seconds between sending PSNP at this level. PSNP interval is not supported.

Syntax

```
int isis_set_circ_level_psnp_interval (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t val);
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
val	PSNP interval.

Output Parameters

None

Return Value

ISIS_API_SET_SUCCESS for set complete.

ISIS_API_SET_ERROR for set not complete; e.g., IS level index not found in the specified circuit index.

isis_get_circ_level_psnp_interval

This call gets the minimum interval in seconds between sending PSNP at this level. PSNP interval switch is not supported.

Syntax

```
int
isis_get_circ_level_psnp_interval (u_int32_t vr_id,
                                   u_int32_t instance, u_int32_t circindex,
                                   u_int32_t level, u_int32_t *ret)
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	PSNP interval. 2 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for supported IS level index not found in the specified circuit index.

isis_get_next_circ_level_metric

This call gets the metric value of this circuit for the next level.

Syntax

```
int isis_get_next_circ_level_metric (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Sub-range for default metric for single hop which picks between 0 to 63.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_wide_metric

This call gets the wide metric value of this circuit for the next level.

Syntax

```
int isis_get_next_circ_level_wide_metric (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Wide metric for IS neighbors which pick between 0 to 1677215.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_priority

This call gets the priority for becoming LAN designated IS at the next level.

Syntax

```
int isis_get_next_circ_level_priority (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Sub-range for IS-IS priority.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_id_octet

This call gets a one-byte identifier that can be used in protocol packets to identify a circuit for the next level.

Syntax

```
int isis_get_next_circ_level_id_octet (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	One-byte identifier. It can be used in protocol packets to identify a circuit.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_id

This call gets the ID of the circuit allocated during initialization for the next level.

Syntax

```
int isis_get_next_circ_level_id (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, int indexlen, struct isis_dis_id *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the ID for a circuit.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_dis

This call gets the ID of the LAN designated IS on this circuit at the next level.

Syntax

```
int isis_get_next_circ_level_dis (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, int indexlen, struct isis_dis_id **ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the LAN designated IS ID.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index, and IF type is Broadcast.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_hello_multiplier

This call gets the hello multiplier that is multiplied by the corresponding HelloTimer, and the result, in seconds (rounded up), is used as the holding time in transmitted hellos, to be used by receivers of hello packets from this IS.

Syntax

```
int isis_get_next_circ_level_hello_multiplier (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Hello multiplier.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_hello_timer

This call gets the next ISIS circ level hello time period, in milliseconds, between IIH PDUs on multiaccess networks at the next level for LANs. The value at level 1 is used as the period between Hellos on L1L2 point-to-point circuits.

Syntax

```
int isis_get_next_circ_level_hello_timer (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Period, in milliseconds, between IIH PDUs on multiaccess networks at this level for LANs. The value at level 1 is used as the period between Hellos on L1L2 point to point circuits.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_dis_hello_timer

This call gets the period, in milliseconds, between hello PDUs on multiaccess networks when this is the designated IS.

Syntax

```
int isis_get_next_circ_level_dis_hello_timer (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	Hello timer of designated IS.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_lsp_throttle

This call gets the minimal interval, in milliseconds, between transmissions of LSPs on an interface at the next level.

Syntax

```
int isis_get_next_circ_level_lsp_throttle (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	LSP minimum interval.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_min_lsp_retrans

This call gets the minimum interval, in seconds, between re-transmission of an LSP at the next level.

Syntax

```
int isis_get_next_circ_level_min_lsp_retrans (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	Minimum LSP re-transmission interval.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_csnp_interval

This call gets the interval, in seconds, between transmission of CSNPs on multiaccess networks if this router is the designated IS at the next level.

Syntax

```
int isis_get_next_circ_level_csnp_interval (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	CSNP interval.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_next_circ_level_psnp_interval

This call gets the minimum interval in seconds between sending PSNP at the next level.

Syntax

```
int isis_get_next_circ_level_psnp_interval (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	PSNP interval. 2 is returned by default.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified circuit index.

ISIS_API_GET_ERROR for the next IS level index not found in the specified circuit index.

isis_get_sys_stat_corrupted_lsps

This call gets the number of corrupted in-memory LSPs detected.

Syntax

```
int isis_get_sys_stat_corrupted_lsps (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of corrupted in-memory LSPs detected.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_auth_type_fails

This call gets the number of authentication type mismatches.

Syntax

```
int isis_get_sys_stat_auth_type_fails (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```


Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of authentication type mismatches.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_auth_fails

This call gets the number of authentication failures.

Syntax

```
int isis_get_sys_stat_auth_fails (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of authentication failures.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_lspdb_overloaded

This call gets the number of times the LSP database has become overloaded.

Syntax

```
int isis_get_sys_stat_lspdb_overloaded (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of times the LSP database has become overloaded.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_man_addr_drop_area

This call gets the number of times a manual address has been dropped from the area.

Syntax

```
int isis_get_sys_stat_man_addr_drop_area (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of times a manual address has been dropped from the area.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_exceed_max_seqnums

This call gets the number of times the IS has attempted to exceed the maximum sequence number.

Syntax

```
int isis_get_sys_stat_exceed_max_seqnums (u_int32_t vr_id, u_int32_t instance,
u_int32_t level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of times IS attempted to exceed the max sequence number.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_seqnum_skips

This call gets the number of times a sequence number skip has occurred.

Syntax

```
int isis_get_sys_stat_seqnum_skips (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	Number of times a sequence number skip has occurred.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_lsp_purges

This call gets the number of times a zero-aged copy of the system's own LSP is received from another node.

Syntax

```
int isis_get_sys_stat_lsp_purges (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	Number of times a zero-aged copy of the system's own LSP is received from some other node.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_id_len_mismatches

This call gets the number of times a PDU is received with a different value for ID field length to that of the receiving system.

Syntax

```
int isis_get_sys_stat_id_len_mismatches (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	Number of times a PDU is received with a different value for ID field length to that of the receiving system.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_max_area_addr_mismatches

This call gets the number of times a PDU is received with a different value for MaximumAreaAddresses from that of the receiving system.

Syntax

```
int isis_get_sys_stat_max_area_addr_mismatches (u_int32_t vr_id, u_int32_t instance,
u_int32_t level,u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

ret	Number of times a PDU is received with a different value for MaximumAreaAddresses from that of the receiving system.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_partition_changes

This call gets the number of times partition changes occurred.

Syntax

```
int isis_get_sys_stat_partition_changes (u_int32_t vr_id, u_int32_t instance, u_int32_t
level, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of times partition changes occurred.

Return Values

ISIS_API_GET_SUCCESS for supported IS level index found in the specified instance.

ISIS_API_GET_ERROR for supported IS level index not found in the specified instance.

isis_get_sys_stat_spf_runs

This call gets the number of times SPF ran at this level.

Syntax

```
int isis_get_sys_stat_spf_runs (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameters

`ret`	Number of times SPF ran at the next level.

Return Value

ISIS_API_GET_SUCCESS for the IS level index found in the specified instance.

ISIS_API_GET_ERROR for the IS level index not found in the specified instance.

isis_get_next_sys_stat_corrupted_lsps

This call gets the number of corrupted in-memory LSPs detected in the next IS system.

Syntax

```
int isis_get_next_sys_stat_corrupted_lsps (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of corrupted in-memory LSPs detected.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_auth_type_fails

This call gets the number of authentication type mismatches in the next IS system.

Syntax

```
int isis_get_next_sys_stat_auth_type_fails (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of authentication type mismatches.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_auth_fails

This call gets the number of authentication failures in the next IS system.

Syntax

```
int isis_get_next_sys_stat_auth_fails (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of authentication failures.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_lspdb_overloaded

This call gets the number of times the LSP database has become overloaded in the next IS system.

Syntax

```
int isis_get_next_sys_stat_lspdb_overloaded (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times the LSP database has become overloaded.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_man_addr_drop_area

This call gets the number of times a manual address has been dropped from the area in the next IS system.

Syntax

```
int isis_get_next_sys_stat_man_addr_drop_area (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times a manual address has been dropped from the area.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_exceed_max_seqnums

This call gets the number of times the next IS has attempted to exceed the maximum sequence number.

Syntax

```
int isis_get_next_sys_stat_exceed_max_seqnums (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times the IS has attempted to exceed the maximum. sequence number.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_seqnum_skips

This call gets the number of times a sequence number skip has occurred in the next IS system.

Syntax

```
int isis_get_next_sys_stat_seqnum_skips (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times a sequence number skip has occurred.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_lsp_purges

This call gets the number of times a zero-aged copy of the system's own LSP is received from some other node in the next IS system.

Syntax

```
int isis_get_next_sys_stat_lsp_purges (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times a zero-aged copy of the system's own LSP is received from some other node.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_id_len_mismatches

This call gets the number of times a PDU is received with a different value for ID field length to that of the receiving system in the next IS system.

Syntax

```
int isis_get_next_sys_stat_id_len_mismatches (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times a PDU is received with a different value for ID field length to that of the receiving system.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_max_area_addr_mismatches

This call gets the number of times a PDU is received with a different value for `MaximumAreaAddresses` from that of the receiving system in the next IS system.

Syntax

```
int isis_get_next_sys_stat_max_area_addr_mismatches (u_int32_t vr_id, u_int32_t
*instance, u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	Number of times a PDU is received with a different value for MaximumAreaAddresses from that of the receiving system.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_partition_changes

This call gets the number of times partition changes occurred in the next IS system.

Syntax

```
int isis_get_next_sys_stat_partition_changes (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *level, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.
indexlen	Length of the index.

Output Parameters

ret	Number of times partition changes occurred.

Return Values

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_next_sys_stat_spf_runs

This call gets the number of times we ran SPF at the next level.

Syntax

```
int isis_get_sys_stat_spf_runs (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level index, that is, 1 for level 1 IS or 2 for level 2 IS.

Output Parameter

`ret`	Number of times SPF ran at the next level.

Return Value

ISIS_API_GET_SUCCESS for the next IS level index found in the specified instance.

ISIS_API_GET_ERROR for the next IS level index not found in the specified instance.

isis_get_circ_adj_changes

This call gets the number of times an adjacency stat change has occurred on this circuit.

Syntax

```
int isis_get_circ_adj_changes (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.

Output Parameters

`ret`	Number of times adjacency state change.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_num_adj

This call gets the number of adjacencies on this circuit.

Syntax

```
int isis_get_circ_adj_changes (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.

circtype IS-IS circuit type.

Output Parameters

ret Number of adjacencies.

Return Value

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_init_fails

This call gets the number of times initialization of this circuit has failed.

Syntax

```
int isis_get_circ_init_fails (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
circtype	IS-IS circuit type.

Output Parameters

ret Number of times initialization of this circuit has failed.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_rej_adjs

This call gets the number of times an adjacency has been rejected on this circuit.

Syntax

```
int isis_get_circ_rej_adjs (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
circtype	IS-IS circuit type.

Output Parameters

ret Number of times an adjacency has been rejected on this circuit.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_id_len_mismatches

This call gets the number of times an IS-IS control PDU with an ID field length different from that of this system has been received.

Syntax

```
int isis_get_circ_id_len_mismatches (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
circtype	IS-IS circuit type.

Output Parameters

ret	Number of times an IS-IS control PDU with an ID field length different from that for this system has been received.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_max_area_addr_mismatches

This call gets the number of times an IS-IS control PDU with a max area address field different from that of this system has been received.

Syntax

```
int isis_get_circ_max_area_addr_mismatches (u_int32_t vr_id, u_int32_t instance,
u_int32_t circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
circtype	IS-IS circuit type.

Output Parameters

ret	Number of times an IS-IS control PDU with a max area address field different from that for this system has been received.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_auth_type_fails

This call gets the number of times an IS-IS control PDU with an auth type field different from that of this system has been received.

Syntax

```
int isis_get_circ_auth_type_fails (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.

Output Parameters

`ret`	Number of times an IS-IS control PDU with an auth type field different from that for this system has been received.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_auth_fails

This call gets the number of times an IS-IS control PDU with the correct auth type has failed to pass authentication validation.

Syntax

```
int isis_get_circ_auth_fails (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.

Output Parameters

`ret`	Number of times an IS-IS control PDU with the correct auth type has failed to pass authentication validation.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_circ_lan_dis_changes

This call gets the number of times the designated IS has changed on this circuit at this level.

Syntax

```
int isis_get_circ_lan_dis_changes (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t circtype, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.

Output Parameters

`ret`	Number of times the designated IS has changed on this circuit at this level. If the circuit is point to point, this count is zero.

Return Values

ISIS_API_GET_SUCCESS for supported circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for supported circuit type not found in the specified circuit index.

isis_get_next_circ_adj_changes

This call gets the number of times an adjacency stat change has occurred on the next circuit level.

Syntax

```
int isis_get_next_circ_adj_changes (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times of adjacency state change.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_num_adj

This call gets the next number of adjacencies on the next circuit level.

Syntax

```
int isis_get_next_circ_num adj (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of adjacencies.

Return Value

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_init_fails

This call gets the number of times initialization of this circuit has failed on the next circuit level.

Syntax

```
int isis_get_next_circ_init_fails (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times initialization of this circuit has failed.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_rej_adjs

This call gets the number of times an adjacency has been rejected on the next circuit level.

Syntax

```
int isis_get_next_circ_rej_adjs (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times an adjacency has been rejected on this circuit.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_id_len_mismatches

This call gets the number of times an IS-IS control PDU with an ID field length different from that for this system has been received on the next circuit level.

Syntax

```
int isis_get_next_circ_id_len_mismatches (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times an IS-IS control PDU with an ID field length different from that of this system has been received.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_max_area_addr_mismatches

This call gets the number of times an IS-IS control PDU with a maximum area address field different from that for this system has been received on the next circuit level.

Syntax

```
int isis_get_next_circ_max_area_addr_mismatches (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t*circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times an IS-IS control PDU with a max area address field different from that of this system has been received.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_auth_type_fails

This call gets the number of times an IS-IS control PDU with an auth type field different from that for this system has been received on the next circuit level.

Syntax

```
int isis_get_next_circ_auth_type_fails (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times an IS-IS control PDU with an auth type field different from that of this system has been received.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_auth_fails

This call gets the number of times an IS-IS control PDU with the correct auth type has failed to pass authentication validation on the next circuit level.

Syntax

```
int isis_get_next_circ_auth_fails (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times an IS-IS control PDU with the correct auth type has failed to pass authentication validation.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_next_circ_lan_dis_changes

This call gets the number of times the designated IS has changed on this circuit at the next level.

Syntax

```
int isis_get_next_circ_lan_dis_changes (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *circtype, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`circtype`	IS-IS circuit type.
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of times the designated IS has changed on this circuit at the next level. If the circuit is point to point, this count is zero.

Return Values

ISIS_API_GET_SUCCESS for the next circuit type found in the specified circuit index.

ISIS_API_GET_ERROR for the next circuit type not found in the specified circuit index.

isis_get_packet_count_hello

This call gets the number of IS-IS Hello PDUs seen in this direction at this level.

Syntax

```
int
isis_get_packet_count_hello (u_int32_t vr_id,
                             u_int32_t instance, u_int32_t circindex,
                             u_int32_t level, u_int32_t direction,
                             u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving

Output Parameters

`ret`	Number of IS-IS Hello PDUs.

Return Values

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_is_hello

This call gets the number of ES-IS Hello PDUs seen in this direction at this level.

Syntax

```
int
isis_get_packet_count_is_hello (u_int32_t vr_id,
                                u_int32_t instance, u_int32_t circindex,
```

```
u_int32_t level, u_int32_t direction,
u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving

Output Parameter

`ret`	Number of ES-IS Hello PDUs. Output is always 0.

Return Value

ISIS_API_ GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_es_hello

This call gets the number of ES Hello PDUs seen in this direction at this level.

Syntax

```
int isis_get_packet_count_es_hello (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t direction, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving

Output Parameter

`ret`	Number of ES Hello PDUs. Output is always 0.

Return Value

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_lsp

This call gets the number of IS-IS LSPs seen in this direction at this level.

Syntax

```
int isis_get_packet_count_lsp (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t direction, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index:
1	Level1
2	Level2
direction	Packet direction:
1	Sending
2	Receiving

Output Parameter

ret	Number of IS-IS LSPs.

Return Values

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_csnp

This call gets the number of IS-IS CSNPs seen in this direction at this level.

Syntax

```
int isis_get_packet_count_csnp (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex,u_int32_t level,u_int32_t direction, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index:
1	Level1

2	Level2
`direction`	Packet direction:
1	Sending
2	Receiving

Output Parameter

`ret`	Number of IS-IS CSNPs.

Return Values

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_psnp

This call gets the number of IS-IS PSNPs seen in this direction at this level.

Syntax

```
int isis_get_packet_count_psnp (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex,u_int32_t level, u_int32_t direction, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
1	Level1
2	Level2
`direction`	Packet direction:
1	Sending
2	Receiving

Output Parameter

`ret`	Number of IS-IS PSNPs.

Return Values

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_packet_count_unknown

This call gets the number of unknown IS-IS PDUs seen in this direction at this level.

Syntax

```
int isis_get_packet_count_unknown (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t level, u_int32_t direction, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving

Output Parameter

`ret`	Number of unknown IS-IS PDUs.

Return Value

ISIS_API_GET_SUCCESS for the packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the packet direction not found in the specified IS-IS level.

isis_get_next_packet_count_hello

This call gets the next number of IS-IS Hello PDUs seen in this direction at this level for the next entry.

Syntax

```
int isis_get_next_packet_count_hello (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, u_int32_t *direction, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of IS-IS Hello PDUs for the next entry.

Return Values

ISIS_API_GET_SUCCESS when the next entry packet direction is found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction not found in the specified IS-IS level.

isis_get_next_packet_count_is_hello

This call gets the next the number of ES-IS Hello PDUs seen in the direction at this level for the next entry.

Syntax

```
int isis_get_next_packet_count_is_hello (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, u_int32_t *direction, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving

Output Parameter

`ret`	Number of ES-IS Hello PDUs for the next entry. Output is always 0.

Return Value

ISIS_API_GET_SUCCESS when the next entry packet direction is found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction is not found in the specified IS-IS level.

isis_get_next_packet_count_es_hello

This call gets the next the number of ES Hello PDUs seen in the direction at this level for the next entry.

Syntax

```
int isis_get_next_packet_count_es_hello (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *level, u_int32_t *direction, u_int32_t *ret);
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	IPacket direction:

1	Sending
2	Receiving

Output Parameter

`ret`	Number of ES Hello PDUs for the next entry. Output is always 0.

Return Value

ISIS_API_GET_SUCCESS for the next entry packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR for the next packet entry direction not found in the specified IS-IS level.

isis_get_next_packet_count_lsp

This call gets the next the number of IS-IS LSPs seen in this direction at this level for the next entry.

Syntax

```
int isis_get_next_packet_count_lsp (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *level, u_int32_t *direction, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
1	Level1
2	Level2
`direction`	Packet direction:
1	Sending
2	Receiving

Output Parameter

`ret`	Number of IS-IS LSPs for the next entry.

Return Values

ISIS_API_GET_SUCCESS when the next entry packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction not found in the specified IS-IS level.

isis_get_next_packet_count_csnp

This call gets the next the number of IS-IS CSNPs seen in this direction at this level for the next entry.

Syntax

```
int
isis_get_next_packet_count_csnp (u_int32_t vr_id,
                                  u_int32_t *instance, u_int32_t *circindex,
                                  u_int32_t *level, u_int32_t *direction,
```

```
                                    int indexlen, u_int32_t *ret)
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index:
1	Level1
2	Level2
direction	Packet direction:
1	Sending
2	Receiving
indexlen	Length of the index.

Output Parameters

ret	Number of IS-IS CSNPs for the next entry.

Return Values

ISIS_API_GET_SUCCESS when the next entry packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction not found in the specified IS-IS level.

isis_get_next_packet_count_psnp

This call gets the next the number of IS-IS PSNPs seen in this direction at this level for the next entry.

Syntax

```
int
isis_get_next_packet_count_psnp (u_int32_t vr_id,
                                 u_int32_t *instance, u_int32_t *circindex,
                                 u_int32_t *level, u_int32_t *direction,
                                 int indexlen, u_int32_t *ret)
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
level	IS-IS level index:
1	Level1
2	Level2
direction	Packet direction:
1	Sending
2	Receiving
indexlen	Length of the index.

Output Parameters

`ret`	Number of IS-IS PSNPs for the next entry.

Return Values

ISIS_API_GET_SUCCESS when the next entry packet direction found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction not found in the specified IS-IS level.

isis_get_next_packet_count_unknown

This call gets the next the number of unknown IS-IS PDUs seen in the next direction at this level for the next entry.

Syntax

```
int
isis_get_next_packet_count_unknown (u_int32_t vr_id,
                                    u_int32_t *instance, u_int32_t *circindex,
                                    u_int32_t *level, u_int32_t *direction,
                                    int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`level`	IS-IS level index:
`1`	Level1
`2`	Level2
`direction`	Packet direction:
`1`	Sending
`2`	Receiving
`indexlen`	Length of the index.

Output Parameters

`ret`	Number of unknown IS-IS PDUs for the next entry.

Return Value

ISIS_API_GET_SUCCESS when the next entry packet direction is found in the specified IS-IS level.

ISIS_API_GET_ERROR when the next entry packet direction is not found in the specified IS-IS level.

isis_get_is_adj_state

This call gets the state of the adjacency.

Syntax

```
int
isis_get_is_adj_state (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
                       u_int32_t adjindex, u_int32_t *ret)
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.

Output Parameters

ret	State of the adjacency.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_3way_state

This call gets the three-way state of the adjacency.

Syntax

```
int isis_get_is_adj_3way_state (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.

Output Parameters

ret	Three-way state of the adjacency.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_nbr_snpa_addr

This call gets the SNPA address of the neighboring IS.

Syntax

```
int isis_get_is_adj_nbr_snpa_addr (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_char **ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.

`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Pointer to the binary SNPA address of the neighboring IS.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_nbr_sys_type

This call gets the type of the neighboring IS.

Syntax

```
int isis_get_is_adj_nbr_sys_type (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Type of the neighboring IS:
`1`	Level 1 intermediate system
`2`	Level 2 intermediate system
`3`	Level 1 and L2 intermediate system on a point-to-point circuit

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_extended_circ_id

This call gets the four byte extended circuit ID learned from the Neighbor during 3-way handshake, or 0.

Syntax

```
int
isis_get_is_adj_extended_circ_id (u_int32_t vr_id, u_int32_t instance,
                                  u_int32_t circindex, u_int32_t adjindex,
                                  u_int32_t *ret);
```

Input Parameter

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Four-byte extended circuit ID learned from the neighbor during 3-way handshake or 0. Output is always 0.

Return Value

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_nbr_sys_id

This call gets the system ID of the neighboring IS.

Syntax

```
int isis_get_is_adj_nbr_sys_id (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_char **ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Pointer to the system ID of the neighboring IS.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_usage

This call gets the adjacency usage with the neighboring IS.

Syntax

```
int isis_get_is_adj_usage (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Adjacency usage with the neighboring IS. Level1 is used for level 1 traffic only. An adjacency of type level2 is used for level 2 traffic only. An adjacency of type level1and2 is used for both level 1 and level 2 traffic on a point-to-point link. There may be two adjacencies (of types level1 and level2) between the same pair of ISs.
`1`	Level1
`2`	Level2
`3`	Level1and2

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_hold_time

This call gets the holding time in seconds for this adjacency.

Syntax

```
int isis_get_is_adj_hold_time (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Holding time in seconds for this adjacency. This value is based on received IIH PDUs and the elapsed time since receipt.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_nbr_priority

This call gets the priority of the neighboring IS for becoming the designated IS.

Syntax

```
int isis_get_is_adj_nbr_priority (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Priority of the neighboring IS for becoming the designated IS.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_is_adj_uptime

This call gets the amount of time in seconds since this adjacency entered 'up'.

Syntax

```
int isis_get_is_adj_uptime (u_int32_t vr_id, u_int32_t instance, u_int32_t circindex,
u_int32_t adjindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

Output Parameters

`ret`	Amount of time in seconds since this adjacency entered state 'up' if the adjacency is up. If the adjacency is not up, the number of seconds since the adjacency was up, or zero, if the adjacency has never been up since the system started.

Return Values

ISIS_API_GET_SUCCESS for the adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the adjacent index not found in the specified circuit.

isis_get_next_is_adj_state

This call gets the state of the next adjacency.

Syntax

```
int isis_get_next_is_adj_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

indexlen	Length of the index.

Output Parameters

ret	State of the next adjacency.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_3way_state

This call gets the next the 3-way state of the next adjacency.

Syntax

```
int isis_get_next_is_adj_3way_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.
indexlen	Length of the index.

Output Parameters

ret	State of the next adjacency.

Return Value

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_nbr_snpa_addr

This call gets the next the SNPA address of the next adjacency.

Syntax

```
int isis_get_next_is_adj_nbr_snpa_addr (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_char **ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.
indexlen	Length of the index.

Output Parameters

`ret`	Pointer to the binary SNPA address of the next neighboring IS.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_nbr_sys_type

This call gets the next the type of the next adjacency.

Syntax

```
int isis_get_next_is_adj_nbr_sys_type (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Type of the next neighboring IS:
`1`	Level 1 intermediate system
`2`	Level 2 intermediate system
`3`	Level 1 and L2 intermediate system on a point-to-point circuit

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_extended_circ_id

This call gets the next the four-byte Extended Circuit ID learned from the Neighbor during 3-way handshake or 0.

Syntax

```
int
isis_get_next_is_adj_extended_circ_id (u_int32_t vr_id, u_int32_t *instance,
                                        u_int32_t *circindex,
                                        u_int32_t *adjindex,
                                        int indexlen, u_int32_t *ret);
```

Input Parameter

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

`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Four-byte extended circuit ID learned from the neighbor during 3-way handshake or 0. Output is always 0.

Return Value

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_nbr_sys_id

This call gets the next the system ID and 4-byte circuit ID of the next adjacency.

Syntax

```
int
isis_get_next_is_adj_nbr_sys_id (u_int32_t vr_id, u_int32_t *instance,
                                 u_int32_t *circindex, u_int32_t *adjindex,
                                 int indexlen, u_char **ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the system ID of the next neighboring IS.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_usage

This call gets the next an adjacency usage of the next adjacency.

Syntax

```
int
isis_get_next_is_adj_usage (u_int32_t vr_id, u_int32_t *instance,
                            u_int32_t *circindex, u_int32_t *adjindex,
                            int indexlen, u_int32_t *ret);
```


Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Usage of the neighboring IS. level1 is used for level 1 traffic only. An adjacency of type level2 is used for level 2 traffic only. An adjacency of type level1and2 is used for both level 1 and level 2 traffic on a point-to-point link. There may be two adjacencies (of types level1 and level2) between the same pair of IS.
`1`	Level1
`2`	Level2
`3`	Level1and2

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_hold_time

This call gets the next the holding time, in seconds, for the next adjacency.

Syntax

```
int isis_get_next_is_adj_hold_time (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`indexlen`	Length of the index.

Output Parameters

`ret`	Holding time in seconds for the next adjacency. This value is based on received IIH PDUs and the elapsed time since receipt.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_nbr_priority

This call gets the next the priority of the next adjacency for becoming the designated IS.

Syntax

```
int isis_get_next_is_adj_nbr_priority (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.
indexlen	Length of the index.

Output Parameters

ret	Priority of the next neighboring IS for becoming the designated IS.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_next_is_adj_uptime

This call gets the next the amount of time, in seconds, since the next adjacency entered 'up'.

Syntax

```
int isis_get_next_is_adj_uptime (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, int indexlen, u_int32_t *ret);
```

Input Parameters

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
circindex	IS-IS circuit index.
adjindex	IS-IS adjacent index.
indexlen	Length of the index.

Output Parameters

ret	Amount of time in seconds since the next adjacency entered state 'up' if the adjacency is up. If the adjacency is not up, the number of seconds since the adjacency was up, or zero if the adjacent has never been up since the system started.

Return Values

ISIS_API_GET_SUCCESS for the next adjacent index found in the specified circuit.

ISIS_API_GET_ERROR for the next adjacent index not found in the specified circuit.

isis_get_is_adj_area_address

This call gets one area address as reported in IIH PDUs received from the adjacent neighbor.

Syntax

```
int isis_get_is_adj_area_address (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t areaindex, struct isis_area_addr **ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`areaindex`	Area index associated with area address advertised by the adjacent neighbor.

Output Parameters

`ret`	Pointer to one area address as reported in IIH PDUs received from the adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the area index found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the area index not found in the specified adjacent neighbor.

isis_get_next_is_adj_area_address

This call gets the next area address as reported in IIH PDUs received from the adjacent neighbor.

Syntax

```
int
isis_get_next_is_adj_area_address (u_int32_t vr_id, u_int32_t *instance,
                                    u_int32_t *curcuit_id, u_int32_t *adjindex,
                                    u_int32_t *areaindex,
                                    int indexlen, struct isis_area_addr **ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`areaindex`	Area index associated with area address advertised by the adjacent neighbor.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to the next area address as reported in IIH PDUs received from the adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the next area index found in the specified adjacent neighbor;

ISIS_API_GET_ERROR for the next area index not found in the specified adjacent neighbor.

isis_get_is_adj_ip_addr_type

This call gets the type of one IP address as reported in IIH PDUs received from the adjacent neighbor.

Syntax

```
int isis_get_is_adj_ip_addr_type (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t ipindex, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`ipindex`	IP address index associated with IP address advertised by the adjacent neighbor.

Output Parameters

`ret`	type of one IP address as reported in `IIH PDU`s received from the adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the IP address index found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the IP address index not found in the specified adjacent neighbor.

isis_get_is_adj_ip_address

This call gets one IP address as reported in IIH PDUs received from the adjacent neighbor.

Syntax

```
int isis_get_is_adj_ip_address (u_int32_t vr_id, u_int32_t instance, u_int32_t
circindex, u_int32_t adjindex, u_int32_t ipindex, struct prefix *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`ipindex`	IP address index associated with IP address advertised by the adjacent neighbor.

Output Parameters

`ret`	Pointer to prefix structure that contains IP address as reported in IIH PDUs received from the adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the IP address index found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the IP address index not found in the specified adjacent neighbor.

isis_get_next_is_adj_ip_addr_type

This call gets the next the type of one IP address as reported in IIH PDUs received from the next adjacent neighbor.

Syntax

```
int isis_get_next_is_adj_ip_addr_type (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, u_int32_t *ipindex, int indexlen, u_int32_t *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`ipindex`	IP address index associated with IP address advertised by the adjacent neighbor.
`indexlen`	Length of the index.

Output Parameters

`ret`	type of one IP address as reported in IIH PDUs received from the next adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the next IP address index found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the next IP address index not found in the specified adjacent neighbor.

isis_get_next_is_adj_ip_address

This call gets the next one IP address as reported in IIH PDUs received from the next adjacent neighbor.

Syntax

```
int isis_get_next_is_adj_ip_address (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*circindex, u_int32_t *adjindex, u_int32_t *ipindex, int indexlen, struct prefix *ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`ipindex`	IP address index associated with IP address advertised by the adjacent neighbor.
`indexlen`	Length of the index.

Output Parameters

`ret`	Pointer to prefix structure that contains IP address as reported in IIH PDUs received from the next adjacent neighbor.

Return Values

ISIS_API_GET_SUCCESS for the next IP address index found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the next IP address index not found in the specified adjacent neighbor.

isis_get_is_adj_prot_supp_protocol

This call gets the type of network protocol supported by the adjacent neighbor.

Syntax

```
int
isis_get_is_adj_prot_supp_protocol (u_int32_t vr_id, u_int32_t instance,
                                    u_int32_t circindex, u_int32_t adjindex,
                                    u_int32_t protocol, u_int32_t *ret)
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.
`protocol`	Supported protocol advertised by the adjacent neighbor

Output Parameters

`ret`	supported protocol advertised by the adjacent neighbor

Return Values

ISIS_API_GET_SUCCESS for the supported protocol found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the supported protocol not found in the specified adjacent neighbor.

isis_get_next_is_adj_prot_supp_protocol

This call gets the type of network protocol supported by the adjacent neighbor corresponding to the next entry.

Syntax

```
int isis_get_next_is_adj_prot_supp_protocol (u_int32_t vr_id, u_int32_t *instance,
u_int32_t *circindex, u_int32_t *adjindex, u_int32_t *protocol, int indexlen, u_int32_t
*ret);
```

Input Parameters

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`circindex`	IS-IS circuit index.
`adjindex`	IS-IS adjacent index.

`protocol`	Supported protocol advertised by the adjacent neighbor
`indexlen`	Length of the index.

Output Parameters

`ret`	Supported protocol advertised by the adjacent neighbor

Return Values

ISIS_API_GET_SUCCESS for the supported protocol found in the specified adjacent neighbor.

ISIS_API_GET_ERROR for the supported protocol not found in the specified adjacent neighbor.

isis_set_ip_ra_nexthop_type

This call sets the type of the IP nexthop address.

Syntax

```
int isis_set_ip_ra_nexthop_type (u_int32_t vr_id, u_int32_t instance, u_int32_t
raindex, u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`val`	Type of the IP nexthop address.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance, and if the nexthop type is the same as the type already present.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_type

This call sets the type of this IP Reachable Address.

Syntax

```
int isis_set_ip_ra_type (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`val`	Type of this IP Reachable Address. Those of type manual are created by the network manager. Those of type automatic are created through propagation of routing information from another routing protocol.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_type

This call gets the type of this IP Reachable Address.

Syntax

```
int isis_get_ip_ra_type (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.

Output Parameter

`ret`	Type of this IP reachable address. Those of type manual are created by the network manager. Those of type automatic are created through propagation of routing information from another routing protocol.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_exist_state

This call sets the state of this IP Reachable Address.

Syntax

```
int
isis_set_ip_ra_exist_state (u_int32_t vr_id, u_int32_t instance,
                            u_int32_t raindex, u_int32_t type, struct prefix p,
                            u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`val`	State of this IP reachable address

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_exist_state

This call gets the state of this IP Reachable Address.

Syntax

```
int
isis_get_ip_ra_exist_state (u_int32_t vr_id, u_int32_t instance,
                            u_int32_t raindex, u_int32_t type, struct prefix p,
                            u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.

Output Parameter

`ret`	State of this IP reachable address:
`1`	isisRowStatusActive
`2`	isisRowStatusNotInservice

3	isisRowStatusNotReady
4	isisRowStatusCreateAndGo
5	isisRowStatusCreateAndWait
6	isisRowStatusDestroy

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_admin_state

This call set the administrative state of the IP Reachable Address.

Syntax

```
int isis_set_ip_ra_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
val	Administrative state of IP Reachable Address.
1	On
2	Off

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_admin_state

This call gets the administrative state of the IP Reachable Address.

Syntax

```
int isis_get_ip_ra_admin_state (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.

`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.

Output Parameter

`ret`	Administrative state of IP Reachable Address.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_metric

This call sets the metric value for reaching the specified destination over this circuit.

Syntax

```
int isis_set_ip_ra_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`val`	Metric value for reaching the specified destination over this circuit.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance, and if the metric value is the same as that already present.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_metric

This call gets the metric value for reaching the specified destination over this circuit.

Syntax

```
int
isis_get_ip_ra_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
```

```
u_int32_t type, struct prefix p, u_int32_t prefixlen,
u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.

Output Parameter

ret	Metric value for reaching the specified destination over this circuit.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_metric_type

This call sets the type of metric that indicates whether the metric is internal or external.

Syntax

```
int isis_set_ip_ra_metric_type (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
val	Type of metric:
1	Internal
2	External

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance, and if the metric type is the same as that already present.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_metric_type

This call gets the type of metric which indicates whether the metric is internal or external.

Syntax

```
int isis_get_ip_ra_metric_type (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.

Output Parameter

`ret`	Type of metric:
`1`	Internal
`2`	External

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_set_ip_ra_full_metric

This call sets the wide metric value for reaching the specified destination over this circuit.

Syntax

```
int isis_set_ip_ra_full_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t val)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`val`	Wide metric value for reaching the specified destination over this circuit.

Output Parameter

None

Return Value

ISIS_API_SET_SUCCESS for IP Reachable Address found in the specified instance, and if the value is the same as that already present.

ISIS_API_SET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_full_metric

This call gets the wide metric value for reaching the specified destination over this circuit.

Syntax

```
int isis_get_ip_ra_full_metric (u_int32_t vr_id, u_int32_t instance, u_int32_t raindex,
u_int32_t type, struct prefix p, u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.

Output Parameter

ret	Wide metric value for reaching the specified destination over this circuit.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_snpa_address

This call gets the SNPA Address to which a PDU may be forwarded in order to reach a destination that matches this IP Reachable Address.

Syntax

```
int
isis_get_ip_ra_snpa_address (u_int32_t vr_id, u_int32_t instance,
                             u_int32_t raindex, u_int32_t type,
                             struct prefix p, u_int32_t prefixlen,
                             u_char **ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.

p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.

Output Parameter

ret	Pointer to the SNPA address to which a PDU may be forwarded to reach a destination.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_ip_ra_source_type

This call gets the origin of this route.

Syntax

```
int
isis_get_ip_ra_source_type (u_int32_t vr_id, u_int32_t instance,
                            u_int32_t raindex,
                            u_int32_t type, struct prefix p,
                            u_int32_t prefixlen, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.

Output Parameter

ret	Origin of this route.

Return Value

ISIS_API_GET_SUCCESS for IP Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_type

This call gets the type of the next IP Reachable Address.

Syntax

```
int isis_get_next_ip_ra_type (u_int32_t vr_id, u_int32_t *instance, u_int32_t *raindex,
u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.

`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`indexlen`	Length of the index.

Output Parameter

`ret`	Type of the next IP Reachable Address. Those of type manual are created by the network manager. Those of type automatic are created through propagation of routing information from another routing protocol.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_exist_state

This call gets the next the state of the next IP Reachable Address.

Syntax

```
int isis_get_next_ip_ra_exist_statex (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen,
u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`indexlen`	Length of the index.

Output Parameter

`ret`	State of the next IP Reachable Address.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_admin_state

This call gets the next the administrative state of the next IP Reachable Address.

Syntax

```
int isis_get_next_ip_ra_admin_state (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen,
u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
indexlen	Length of the index.

Output Parameter

ret	Administrative state of the next IP Reachable Address.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_metric

This call gets the next metric value for reaching the next destination over this circuit.

Syntax

```
int isis_get_next_ip_ra_metric (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen,
u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
indexlen	Length of the index.

Output Parameter

None

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_metric_type

This call gets the next type of metric that indicates whether the metric is internal or external.

Syntax

```
int
isis_get_next_ip_ra_metric_type (u_int32_t vr_id, u_int32_t *instance,
                                 u_int32_t *raindex, u_int32_t *type,
                                 struct prefix *p, u_int32_t *prefixlen,
                                 int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`indexlen`	Length of the index.

Output Parameter

`ret`	Type of metric that indicates whether the metric is internal or external.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_full_metric

This call gets the next the wide metric value for reaching the next destination over the circuit.

Syntax

```
int isis_get_next_ip_ra_full_metric (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen,
u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`raindex`	Identifier to specify isisIPRAEntry.
`type`	Type of IP Reachable Address.
`p`	Destination of IP Reachable Address.
`prefixlen`	Length of the IP netmask of IP Reachable Address.
`indexlen`	Length of the index.

Output Parameter

ret	Wide metric value for reaching the next destination over this circuit.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_snpa_address

This call gets the next the SNPA Address to which a PDU may be forwarded in order to reach the specified destination which matches the next IP Reachable Address.

Syntax

```
int isis_get_next_ip_ra_snpa_address (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen, u_char
**ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
indexlen	Length of the index.

Output Parameter

ret	Pointer to the SNPA Address to which a PDU may be forwarded in order to reach the next destination.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_next_ip_ra_source_type

This call gets the next the origin of the next route.

Syntax

```
int isis_get_next_ip_ra_source_type (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*raindex, u_int32_t *type, struct prefix *p, u_int32_t *prefixlen, int indexlen,
u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.

raindex	Identifier to specify isisIPRAEntry.
type	Type of IP Reachable Address.
p	Destination of IP Reachable Address.
prefixlen	Length of the IP netmask of IP Reachable Address.
indexlen	Length of the index.

Output Parameter

ret	Origin of the next route.

Return Value

ISIS_API_GET_SUCCESS for the next Reachable Address found in the specified instance.

ISIS_API_GET_ERROR for the next IP Reachable Address not found in the specified instance.

isis_get_lsp_seq

This call gets the sequence number for this LSP.

Syntax

```
int isis_get_lsp_seq (u_int32_t vr_id, u_int32_t instance, u_int32_t level, struct
isis_lspid lspid, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.
level	IS-IS level this LSP belongs to.
lspid	LSP ID for this LSP.

Output Parameter

ret	Sequence number for this LSP.

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_lsp_zero_life

This call gets the state that indicates whether or not this LSP is being purged by this system.

Syntax

```
int isis_get_lsp_zero_life (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
struct isis_lspid lspid, u_int32_t *ret)
```

Input Parameter

vr_id	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
instance	IS-IS instance ID.

`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.

Output Parameter

`ret`	State indicating whether or not this LSP is being purged by this system:
`1`	Purged
`2`	Not purged

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_lsp_checksum

This call gets the 16-bit fletcher checksum for this LSP.

Syntax

```
int isis_get_lsp_checksum (u_int32_t vr_id, u_int32_t instance, u_int32_t level, struct
isis_lspid lspid, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.

Output Parameter

`ret`	Checksum for this LSP.

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_lsp_lifetime_remain

This call gets the remaining lifetime, in seconds, for this LSP.

Syntax

```
int
isis_get_lsp_lifetime_remain (u_int32_t vr_id, u_int32_t instance,
                              u_int32_t level, struct isis_lspid lspid,
                              u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.

`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.

Output Parameter

`ret`	Remaining lifetime in seconds for this LSP.

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_lsp_pdu_length

This call gets the length of this LSP.

Syntax

```
int
isis_get_lsp_pdu_length (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
                         struct isis_lspid lspid, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.

Output Parameter

`ret`	Length of this LSP.

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_lsp_attributes

This call gets the flags carried by this LSP.

Syntax

```
int
isis_get_lsp_attributes (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
                         struct isis_lspid lspid, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.

`lspid`	LSP ID for this LSP.

Output Parameter

`ret`	Flags carried by this LSP.

Return Value

ISIS_API_GET_SUCCESS for LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for LSP not found in the specified instance at this level.

isis_get_next_lsp_seq

This call gets the next sequence number for the next LSP.

Syntax

```
int isis_get_next_lsp_seq (u_int32_t vr_id, u_int32_t *instance, u_int32_t *level,
struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	Sequence number for the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_next_lsp_zero_life

This call gets the next state indicating whether or not the next LSP is being purged by this system.

Syntax

```
int isis_get_next_lsp_zero_life (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	State indicating whether or not the next LSP is being purged by this system:
`1`	Purged
`2`	Not purged

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_next_lsp_checksum

This call gets the next the 16-bit fletcher checksum for the next LSP.

Syntax

```
int isis_get_next_lsp_checksum (u_int32_t vr_id, u_int32_t *instance, u_int32_t *level,
struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	Checksum for the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_next_lsp_lifetime_remain

This call gets the next remaining lifetime in seconds for the next LSP.

Syntax

```
int isis_get_next_lsp_lifetime_remain (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	Remaining lifetime in seconds for the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_next_lsp_pdu_length

This call gets the next length of the next LSP.

Syntax

```
int isis_get_next_lsp_pdu_length (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	Length of the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_next_lsp_attributes

This call gets the next the flags carried by the next LSP.

Syntax

```
int isis_get_next_lsp_attributes (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`indexlen`	Length of the index.

Output Parameter

`ret`	Flags carried by the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next LSP found in the specified instance at this level.

ISIS_API_GET_ERROR for the next LSP not found in the specified instance at this level.

isis_get_lsp_tlv_index

This call gets the index of this TLV in the LSP. This object follows the index behavior.

Syntax

```
int isis_get_lsp_tlv_index (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
struct isis_lspid lspid, u_int32_t tlvindex, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	Index of this TLV in the LSP.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_lsp_tlv_seq

This call gets the sequence number for this LSP.

Syntax

```
int isis_get_lsp_tlv_seq (u_int32_t vr_id, u_int32_t instance, u_int32_t level, struct
isis_lspid lspid, u_int32_t tlvindex, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	Sequence number for this LSP.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_lsp_tlv_checksum

This call gets the 16-bit Fletcher checksum for this LSP.

Syntax

```
int isis_get_lsp_tlv_checksum (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
struct isis_lspid lspid, u_int32_t tlvindex, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	16-bit Fletcher checksum for this LSP.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_lsp_tlv_type

This call gets the type of this TLV.

Syntax

```
int isis_get_lsp_tlv_type (u_int32_t vr_id, u_int32_t instance, u_int32_t level, struct
isis_lspid lspid, u_int32_t tlvindex, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For a non-VR implementation, pass 0.
`instance`	IS-IS instance ID.
`level`	The IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	The type of this TLV in this LSP.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_lsp_tlv_len

This call gets the length of this TLV.

Syntax

```
int isis_get_lsp_tlv_len (u_int32_t vr_id, u_int32_t instance, u_int32_t level, struct
isis_lspid lspid, u_int32_t tlvindex, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	Length of this TLV in this LSP.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_lsp_tlv_value

This call gets the value of this TLV.

Syntax

```
int isis_get_lsp_tlv_value (u_int32_t vr_id, u_int32_t instance, u_int32_t level,
struct isis_lspid lspid, u_int32_t tlvindex, struct isis_tlv **ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

Output Parameter

`ret`	Pointer to the next TLV structure in this LSP. The “data” member of this structure contains pointer to the TLV value.

Return Value

ISIS_API_GET_SUCCESS for TLV found in the specified LSP.

ISIS_API_GET_ERROR for TLV not found in the specified LSP.

isis_get_next_lsp_tlv_index

This call gets the next index of the next TLV in the LSP. This object follows the index behavior.

Syntax

```
int isis_get_next_lsp_tlv_index (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, u_int32_t *tlvindex, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.
`indexlen`	Length of the index.

Output Parameter

`ret`	Index of the next TLV in the LSP.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

isis_get_next_lsp_tlv_seq

This call gets the next the sequence number for the next LSP.

Syntax

```
int isis_get_next_lsp_tlv_seq (u_int32_t vr_id, u_int32_t *instance, u_int32_t *level,
struct isis_lspid *lspid, u_int32_t *tlvindex, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.

`indexlen` Length of the index.

Output Parameter

`ret` Sequence number for this LSP.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

isis_get_next_lsp_tlv_checksum

This call gets the next 16-bit Fletcher checksum for the next LSP.

Syntax

```
int isis_get_next_lsp_tlv_checksum (u_int32_t vr_id, u_int32_t *instance, u_int32_t
*level, struct isis_lspid *lspid, u_int32_t *tlvindex, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.
`indexlen`	Length of the index.

Output Parameter

`ret` 16-bit Fletcher checksum for the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

isis_get_next_lsp_tlv_type

This call gets the next type of the next TLV.

Syntax

```
int isis_get_next_lsp_tlv_type (u_int32_t vr_id, u_int32_t *instance, u_int32_t *level,
struct isis_lspid *lspid, u_int32_t *tlvindex, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.

`tlvindex`	Index of this TLV.
`indexlen`	Length of the index.

Output Parameter

`ret`	Type of this TLV in the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

isis_get_next_lsp_tlv_len

This call gets the next length of the next TLV.

Syntax

```
int
isis_get_next_lsp_tlv_len (u_int32_t vr_id, u_int32_t *instance,
                           u_int32_t *level, struct isis_lspid *lspid,
                           u_int32_t *tlvindex, int indexlen, u_int32_t *ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.
`indexlen`	Length of the index.

Output Parameter

`ret`	Length of this TLV in the next LSP.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

isis_get_next_lsp_tlv_value

This call gets the next value of the next TLV.

Syntax

```
int
isis_get_next_lsp_tlv_value (u_int32_t vr_id, u_int32_t *instance,
                             u_int32_t *level, struct isis_lspid *lspid,
                             u_int32_t *tlvindex, int indexlen,
                             struct isis_tlv **ret)
```

Input Parameter

`vr_id`	Virtual Router ID. The default value is 0. For non-VR implementation, pass 0 for vr_id.
`instance`	IS-IS instance ID.
`level`	IS-IS level this LSP belongs to.
`lspid`	LSP ID for this LSP.
`tlvindex`	Index of this TLV.
`indexlen`	Length of the index.

Output Parameter

`ret`	Pointer to the next TLV structure in this LSP. The “data” member of this structure contains pointer to the TLV value.

Return Value

ISIS_API_GET_SUCCESS for the next TLV found in the specified LSP.

ISIS_API_GET_ERROR for the next TLV not found in the specified LSP.

CHAPTER 11 IS-IS SNMP Traps

This chapter includes a description and definition of each IS-IS SNMP trap.

isisAdjacencyChange

This variable sends a notification when an adjacency changes states, including whether it is entering or leaving a state.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisAdjacencyChange
     OBJECTS {
       isisSystemLevel,
       isisCircIfIndex,
       isisTrapLSPID,
       isisAdjState }
     ::= { isisTrapPrefix 17 } */

void
isisAdjacencyChange (struct isis_nbr_level *nl)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISADJACENCYCHANGE;
  u_int32_t instance = nl->ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (nl->index);
  u_int32_t circindex = nl->ifl->isi->circuit_id;
  u_int32_t adjindex = nl->adjacency_id;
  u_int32_t ifindex, adjstate;
  u_int32_t vr_id = 0;
  u_char *source_id;
  struct isis_lspid lspid;
  struct isis_master *im = nl->ifl->isi->top->im;
```

isisAreaMismatch

This variable sends a notification when a Hello PDU from an IS that does not share an area address is received. This notification includes the header of the packet, which helps identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source. This decision is up to the agent to make, and may be based on the circuit or on certain MAC level information.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisAreaMismatch
     OBJECTS {
       isisSystemInstance,
```

```
        isisSystemLevel,
        isisCircIfIndex,
        isisPDUFragment }
      ::= { isisTrapPrefix 12 } */

void
isisAreaMismatch (struct isis_if_level *ifl, struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISAREAMISMATCH;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisAttemptToExceedMaxSequence

This notification is sent when the sequence number on a generated LSP wraps the 32-bit sequence counter. ZebOS-XP purges and waits to re-announce this information, and will generate an event each time this action happens.

Definition

```
 isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisAttemptToExceedMaxSequence
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisTrapLSPID }
     ::= { isisTrapPrefix 4 } */

void
isisAttemptToExceedMaxSequence (struct isis_lsp *lsp)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[3];
  int spec_trap_val = ISISATTEMPTTOEXCEEDMAXSEQUENCE;
  u_int32_t instance = lsp->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (lsp->il->index);
  struct isis_lspid lspid;
  struct isis_master *im = lsp->il->top->im;

  namelen = sizeof (isis_oid) / sizeof (oid);
```


isisAuthenticationFailure

This variable sends a notification when a PDU with an incorrect authentication information field is received. This notification includes the header of the packet, which helps identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification from the same source.

Definition

```
void
isisAuthenticationFailure (struct isis_if_level *ifl, struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISAUTHENTICATIONFAILURE;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisAuthenticationTypeFailure

This variable sends a notification when a PDU with the wrong authentication-type field is received. This notification includes the header of the packet, which helps identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisAuthenticationFailure
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisCircIfIndex,
       isisPDUFragment }
     ::= { isisTrapPrefix 10 } */
void
isisAuthenticationTypeFailure (struct isis_if_level *ifl,
                               struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISAUTHENTICATIONTYPEFAILURE;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisCorruptedLSPDetected

This notification is sent when ZebOS-XP finds that an LSP stored in memory has become corrupted. ZebOS-XP forwards the LSP ID. ZebOS-XP might have independent knowledge of this ID, but the ID might be corrupt.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisCorruptedLSPDetected
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisTrapLSPID }
     ::= { isisTrapPrefix 3 } */
void
isisCorruptedLSPDetected (struct isis_level *il, struct isis_lsp_header *lsph)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[3];
  int spec_trap_val = ISISCORRUPTEDLSPDETECTED;
  u_int32_t instance = il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (il->index);
  struct isis_master *im = il->top->im;

  namelen = sizeof (isis_oid) / sizeof (oid);
```

isisDatabaseOverload

This variable is used when the ZebOS-XP enters or leaves the IS-IS database overload state.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisDatabaseOverload
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisSysLevelOverloadState }
   ::= { isisTrapPrefix 1 } */
void
isisDatabaseOverload (struct isis_level *il)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[3];
  int spec_trap_val = ISISDATABASEOVERLOAD;
  u_int32_t instance = il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (il->index);
  u_int32_t state;
  struct isis_master *im = il->top->im;
  u_int32_t vr_id = 0;
```



```
  /* Get isisSysLevelOverloadState. */
  if (isis_get_sys_level_overload_state (vr_id, instance, level, &state)
      != ISIS_API_GET_SUCCESS)
    return;

  namelen = sizeof (isis_oid) / sizeof (oid);
```

isisIDLenMismatch

This variable sends a notification when a PDU with a different value for the system ID length is received. It includes an index to identify the circuit from where the PDU came from and a header of the PDU to identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification from same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisIDLenMismatch
     OBJECTS {
       isisSystemInstance
       isisFieldLen,
       isisCircIfIndex,
       isisPDUFragment }
     ::= { isisTrapPrefix 5 } */
void
isisIDLenMismatch (struct isis_if_level *ifl, struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISIDLENMISMATCH;
  struct isis_header *ih = (struct isis_header *) STREAM_DATA (pkt->buf);
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t id_length = ih->id_length;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisLSPTooLargeToPropagate

This variable sends a notification about an attempt to propagate an LSP that is larger than the block size of a circuit. This is an edge-triggered notification. ZebOS-XP does not send a second notification received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisLSPTooLargeToPropagate
     OBJECTS {
       isisSystemLevel,
       isisCircIfIndex,
       isisLSPSize,
       isisTrapLSPID }
     ::= { isisTrapPrefix 14 } */
```

```
void
isisLSPTooLargeToPropagate (struct isis_interface *isi, struct isis_lsp *lsp)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISLSPTOOLARGETOPROPAGATE;
  u_int32_t instance = isi->top->instance_id;
  u_int32_t level = INDEX2LEVEL (lsp->il->index);
  u_int32_t circindex = isi->circuit_id;
  u_int32_t size = pal_ntoh16 (lsp->lsph->pdu_length);
  u_int32_t ifindex;
  struct isis_master *im = isi->top->im;
  u_int32_t vr_id = 0;

  /* Get isisCircIfIndex. */
  if (isis_get_circ_ifindex (vr_id, instance, circindex, &ifindex)
      != ISIS_API_GET_SUCCESS)
    return;

  namelen = sizeof (isis_oid) / sizeof (oid);
```

isisManualAddressDrops

This variable is sent when an area address assigned to this system is ignored when computing routes. An object describes the area that is ignored and a variable contains the number of times this event occurs. This notification is edge triggered and does not regenerate until an address that was used in the previous computation was dropped.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisManualAddressDrops
     OBJECTS {
       isisSystemInstance,
       isisManAreaAddrExistState }
     ::= { isisTrapPrefix 2 } */
void
isisManualAddressDrops (struct isis *top, struct isis_area_addr *addr)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[2];
  int spec_trap_val = ISISMANUALADDRESSDROPS;
  u_int32_t instance = top->instance_id;
  u_int32_t state = isisRowStatusDestroy;
  struct isis_master *im = top->im;

  namelen = sizeof (isis_oid) / sizeof (oid);

  /* Set isisSystemInstance. */
  ISIS_SNMP_TRAP_SET_INSTANCE (obj[0], &instance);

  /* Set isisManAreaAddrExistState. */
  ISIS_SNMP_TRAP_SET_AREA_ADDR_STATE (obj[1], instance, addr, &state);
```



```
  /* Call registered trap callbacks. */
  for (index = 0; index < vector_max (im->traps[spec_trap_val - 1]); index++)
    if ((func = vector_slot (im->traps[spec_trap_val - 1], index)))
      (*func) (isis_trap_oid, sizeof (isis_trap_oid) / sizeof (oid),
               spec_trap_val, isis_instance_uptime (im, instance),
               obj, sizeof (obj) / sizeof (struct snmp_trap_object));
}
```

isisMaxAreaAddressesMismatch

This variable sends a notification when a PDU with a different value of the maximum area addresses is received. It includes a header of the packet, which helps identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisMaxAreaAddressesMismatch
     OBJECTS {
       isisSystemInstance,
       isisMaxAreaAddress,
       isisCircIfIndex,
       isisPDUFragment }
     ::= { isisTrapPrefix 6 } */
void
isisMaxAreaAddressesMismatch (struct isis_if_level *ifl,
                              struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISMAXAREAADDRESSESMISMATCH;
  struct isis_header *ih = (struct isis_header *) STREAM_DATA (pkt->buf);
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t area_addresses = ih->max_area_addresses;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisOriginatingLSPBufferSizeMismatch

This variable sends a notification when a level 1 LSP or level 2 LSP is received, which is larger than the value for `isisOriginatingBufferSize`. Or, when an LSP is received containing `isisOriginatingBufferSize` and the value in the PDU option field does not match the local value for `isisOriginatingBufferSize`. ZebOS-XP passes up the size from the option field or the size of the LSP that exceeds a configuration. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisOriginatingLSPBufferSizeMismatch
```

```
      OBJECTS {
        isisSystemLevel,
        isisCircIfIndex,
        isisTrapLSPID,
        isisOriginatingBufferSize }
      ::= isisTrapPrefix 15 } */
void
isisOriginatingLSPBufferSizeMismatch (struct isis_if_level *ifl,
                                      struct isis_lsp_header *lsph,
                                      u_int32_t size)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISORIGINATINGLSPBUFFERSIZEMISMATCH;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisOwnLSPPurge

This variable sends a notification when a PDU with the current system ID and zero age is received. This notification includes the circuit index (if available), which helps identify the source of confusion.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisOwnLSPPurge
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisCircIfIndex,
       isisTrapLSPID }
     ::= { isisTrapPrefix 7 } */
void
isisOwnLSPPurge (struct isis_if_level *ifl, struct isis_lsp *lsp)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISOWNLSPPURGE;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```


isisProtocolsSupportedMismatch

This variable sends a notification when a non-pseudonode segment 0 LSP is received that has no matching protocols supported. This might be because the system does not generate the field or because there are no common elements. The list of protocols supported is included in the notification, which might be empty if the TLV is not supported or if the TLV is empty. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisProtocolsSupportedMismatch
     OBJECTS {
       isisSystemLevel,
       isisCircIfIndex,
       isisProtocolsSupported,
       isisTrapLSPID,
       isisPDUFragment }
     ::= isisTrapPrefix 16 } */
void
isisProtocolsSupportedMismatch (struct isis_if_level *ifl,
                                struct isis_lsp *lsp,
                                u_char proto_type)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[5];
  int spec_trap_val = ISISPROTOCOLSSUPPORTEDMISMATCH;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_char len, proto[2];
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisRejectedAdjacency

This variable sends notification when a hello PDU is received from an IS, but did not form an adjacency. This is an edge-triggered notification. ZebOS-XP does not send a second notification about PDUs received from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisRejectedAdjacency
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisCircIfIndex,
       isisPDUFragment }
     ::= { isisTrapPrefix 13 } */
void
```

```
isisRejectedAdjacency (struct isis_if_level *ifl, struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISREJECTEDADJACENCY;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisSequenceNumberSkip

When ZebOS-XP receives an LSP without a system ID and with different contents, ZebOS-XP might need to reissue the LSP with a higher sequence number. ZebOS-XP sends this notification to increase the sequence number by more than one. Specifically, this notification is used if two intermediate systems are configured with the same system ID.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
   isisSequenceNumberSkip
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisCircIfIndex,
       isisTrapLSPID }
     ::= { isisTrapPrefix 8 } */
void
isisSequenceNumberSkip (struct isis_if_level *ifl, struct isis_lsp *lsp)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[4];
  int spec_trap_val = ISISSEQUENCENUMBERSKIP;
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```

isisVersionSkew

This variable sends a notification when a hello PDU from an IS running a different version of the protocol is received. It includes the header of the packet, that might help identify the source of confusion. This is an edge-triggered notification. ZebOS-XP does not send a second notification from the same source.

Definition

```
/* isisNotifications ::= { isisMIB 2 }
   isisTrapPrefix ::= { isisNotifications 0 }
```



```
   isisVersionSkew
     OBJECTS {
       isisSystemInstance,
       isisSystemLevel,
       isisCircIfIndex,
       isisProtocolVersion,
       isisPDUFragment }
     ::= { isisTrapPrefix 11 } */
void
isisVersionSkew (struct isis_if_level *ifl, struct isis_packet *pkt)
{
  oid *ptr;
  int index, namelen;
  SNMP_TRAP_CALLBACK func;
  struct snmp_trap_object obj[5];
  int spec_trap_val = ISISVERSIONSKEW;
  struct isis_header *ih = (struct isis_header *) STREAM_DATA (pkt->buf);
  u_int32_t instance = ifl->il->top->instance_id;
  u_int32_t level = INDEX2LEVEL (ifl->index);
  u_int32_t circindex = ifl->isi->circuit_id;
  u_int32_t version = ih->version;
  u_int32_t ifindex;
  struct isis_master *im = ifl->isi->top->im;
  u_int32_t vr_id = 0;
```


Index

L

N

O

P

R

S

V