ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Open Shortest Path First
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 Open Shortest Path First (OSPF).

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This guide contains these chapters and appendices:

- Chapter 1, *Introduction*
- Chapter 2, *Graceful Restart*
- Chapter 3, *Optimization*
- Chapter 4, *Multiple Instances*
- Chapter 5, *PE-CE for BGP/MPLS VPNs*
- Chapter 6, *Passive Interface*
- Chapter 7, *Multi-Area Adjacency*
- Chapter 8, *Constrained Shortest Path First*
- Chapter 9, *Loop-Free Alternate Fast Reroute*
- Chapter 10, *On Demand Circuit*
- Chapter 11, *Data Structures and Constants*
- Chapter 12, *OSPFv2 Command API*
- Chapter 13, *OSPFv3 Command API*
- Chapter 14, *OSPFv2 SNMP API*

- Chapter 15, *OSPFv3 SNMP API*
- Appendix A, *Message and TLV Types*
- Appendix B, *Source Code*

Related Documents

The following guides are related to this document:

- *Open Shortest Path First Command Reference*
- *Unicast Configuration Guide*

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

Support

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

Comments

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

CHAPTER 1 Introduction

This chapter covers the fundamentals of the ZebOS-XP implementation of the Open Shortest Path First (OSPF) protocol.

Features

- Graceful Restart
- Constrained Shortest Path First
- Optimization
- Multiple Instances
- PE-CE for BGP/MPLS VPNs
- Passive Interface
- Multi-Area Adjacency
- Loop-Free Alternate Fast Reroute

Overview

OSPF is an interior gateway protocol that was designed for TCP/IP networks to address the scaling issues of distance-vector routing protocols such as RIP.

OSPF was developed by a working group of the IETF (Internet Engineering Task Force). Development started in 1988 and was completed in 1991. Updates to the protocol continue to be published.

As shown in Table 1-1, ZebOS-XP has two separate daemons that support the two major OSPF standards:

Table 1-1: ZebOS-XP OSPF daemons

Version name	Daemon name	Address format	Standard	Year
OSPF version 2	`ospfd`	IPv4	RFC 2328	1998
OSPF version 3	`ospf6d`	IPv6	RFC 5340	2008

Link-State Protocol

OSPF is a link-state protocol. A link state is the description of an interface on a router (for example, IP address, subnet mask, type of network) and its relationship to neighboring routers. The collection of these link states forms a link-state database (LSDB, also called a topology database) which contains the set of link-state advertisements describing the OSPF network and its external connections. Each router within an area maintains an identical copy of the link-state database.

Each OSPF router executes the Shortest-Path First (SPF) algorithm (also called the Djikstra algorithm) to process the information stored in the link state database. The algorithm produces a tree-representation of the network detailing the preferred routes to each destination network. The router running the SPF algorithm is the root of the tree. The output of the algorithm is the list of shortest-paths to each destination network.

Interface Cost

Each OSPF interface (link) has a cost which is, by default, an arbitrary metric calculated using the following formula:

cost = reference-bandwidth (10^8)/ interface bandwidth

You can override the default cost of an interface using a metric based on any criteria important to you such as bandwidth, distance, quality of service, or a combination of these and other factors.

Routes with a lower total path cost are preferred over those with a higher path cost. When several equal-cost routes to a destination exist, traffic is distributed equally among them.

Autonomous System and Areas

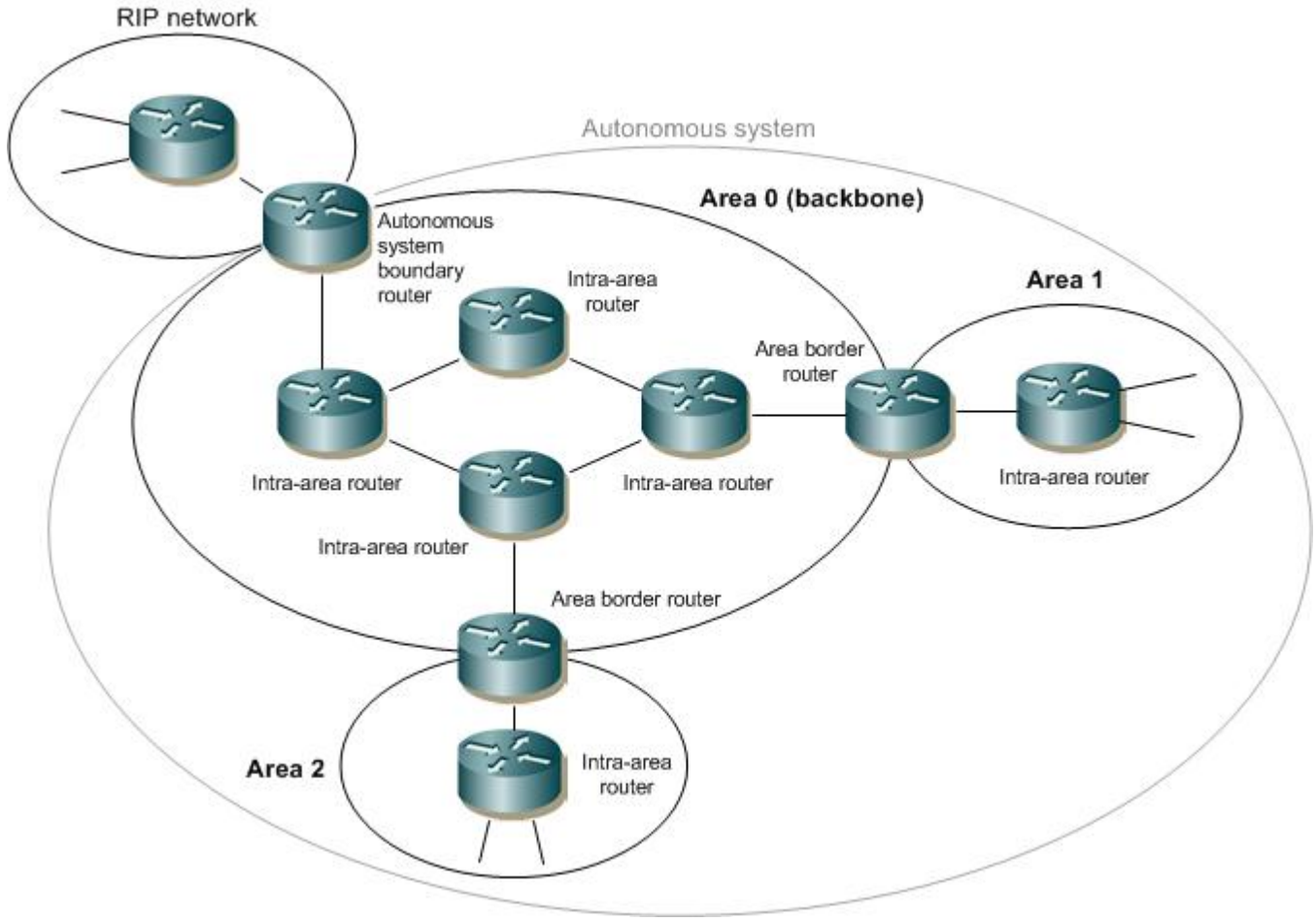


Figure 1-1: OSPF Autonomous System and Areas

As shown in Figure 1-1, all routers under one OSPF process belong to the same OSPF Autonomous System. OSPF breaks the Autonomous System (AS) into smaller domains called areas. An area consists of a logical grouping of networks and routers. Each area has its own link-state database that is duplicated on each router in the same area. A router belonging to two areas maintains two link state databases.

An area can coincide with geographic or administrative boundaries. Each area is assigned an area identifier. Areas isolate traffic and reduce the size of the link-state database. It usually makes the most sense to configure an area as a set of contiguous IP subnetworks.

Backbone Area

All OSPF networks contain at least one area known as area 0 or the backbone area. Additional areas can be created based on network topology or other design requirements.

In networks containing multiple areas, the backbone area is the central area to which all other areas connect. OSPF expects each connected area to announce its summary routing information directly into the backbone. The backbone then announces this information into other areas.

Routing Types

As shown in Figure 1-1, these are the basic types of routing in OSPF:

- Intra-area routing – the source and destination are in the same area
- Inter-area routing – the source and destination are in different areas
- External routing – the source and destination are in different networks and one of the networks can be running a routing protocol other than OSPF such as RIP or BGP

Routers have different tasks in OSPF based on where they reside and how they are configured:

- A backbone router has at least one connection to area 0.
- An Area Border Router (ABR) attaches to area 0 and at least one other area. An ABR maintains a separate database for each attached area and a separate instance of the SPF algorithm for each attached area. The backbone enables the exchange of summary information between area border routers. Every area border router hears the area summaries from all other area border routers.
- An intra-area router has all its connections in only one area. Intra-area routers only maintain a link-state database for their local area.
- An Autonomous System Boundary Router (ASBR) connects to another network domain such as RIP and advertises external routes from that network domain into OSPF. An ASBR is located at the edge of an OSPF network.

Regular Areas

As shown in Figure 1-2, in a regular area, addresses from other areas and addresses from external routes (learned from other routing protocols such as RIP and BGP) are injected as summary addresses. All regular areas must connect to the backbone area.

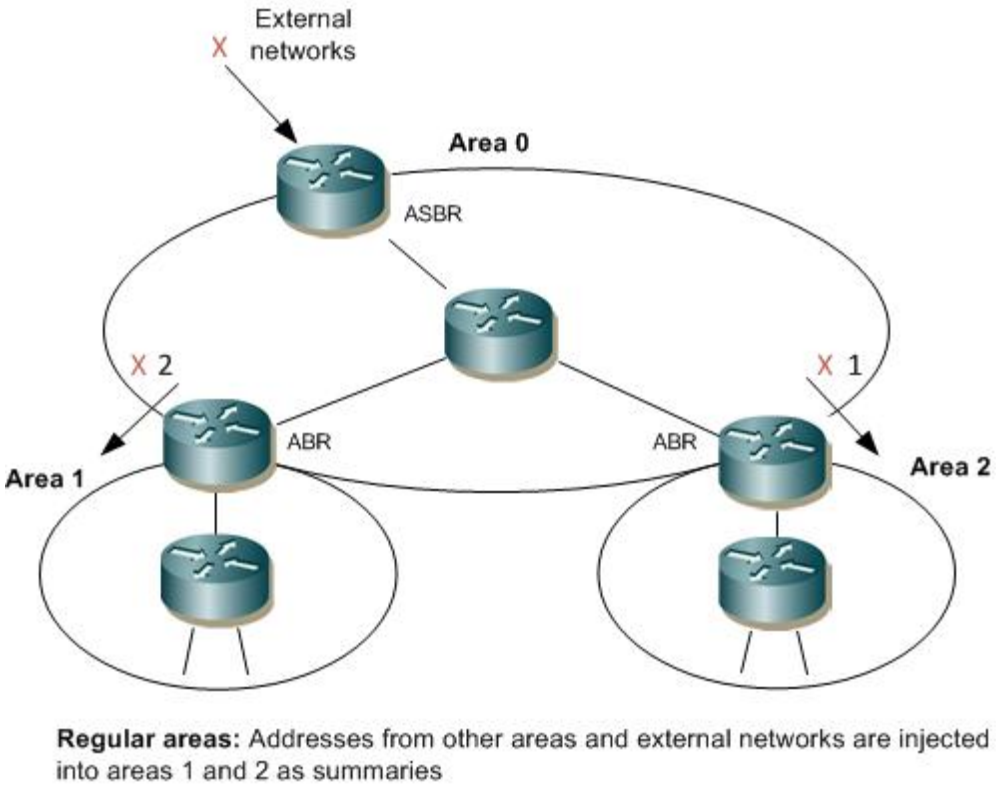


Figure 1-2: Regular Area

Stub Areas

As shown in Figure 1-3, stub areas are shielded from external routes but receive information about networks that belong to other areas of the same OSPF autonomous system. A default summary route (address 0.0.0.0) is inserted into the stub area to reach external routes.

In a stub area, the link state databases contain only the default route and intra-area and inter-area routes. This minimizes the size of the link-state database and memory in the stub area routers.

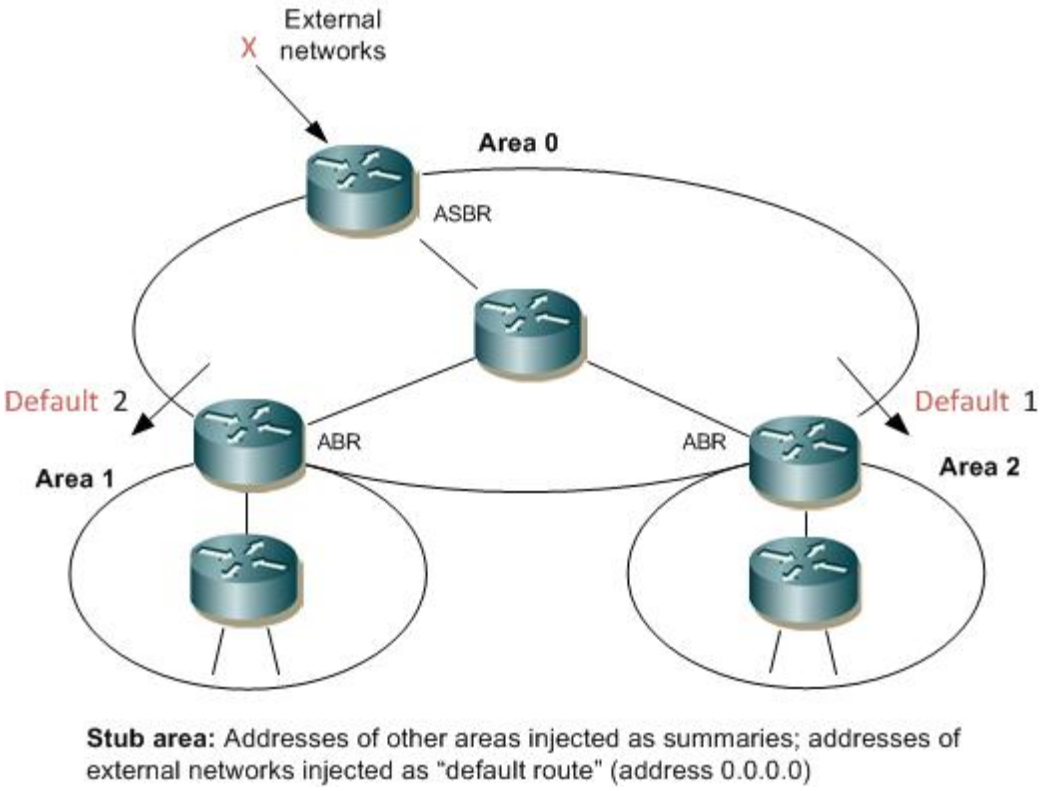


Figure 1-3: Stub Area

A stub area cannot contain an ASBR, so external routes cannot be generated from within the stub area.

All routers within the area must be configured as stub routers. This configuration is verified through the exchange of adjacency packets.

A stub area is at the edge of an OSPF autonomous system and is typically set up in situations where a branch office need not know about all the routes to every other office. Instead it can use a default route to the central office and get to other destinations from there.

Stub areas can be deployed when there is a single exit point connecting the area to the backbone. An area with multiple exit points can also be a stub area. However, there is no guarantee that packets exiting the area will follow an optimal path. This is because each ABR generates a default route. There is no ability to associate traffic with a specific default route.

Totally Stubby Areas

In a totally stubby area as shown in Figure 1-4, routers only maintain information in their link-state database about intra-area routes and a default route. Addresses from both other areas and external networks are injected as a default route (address 0.0.0.0).

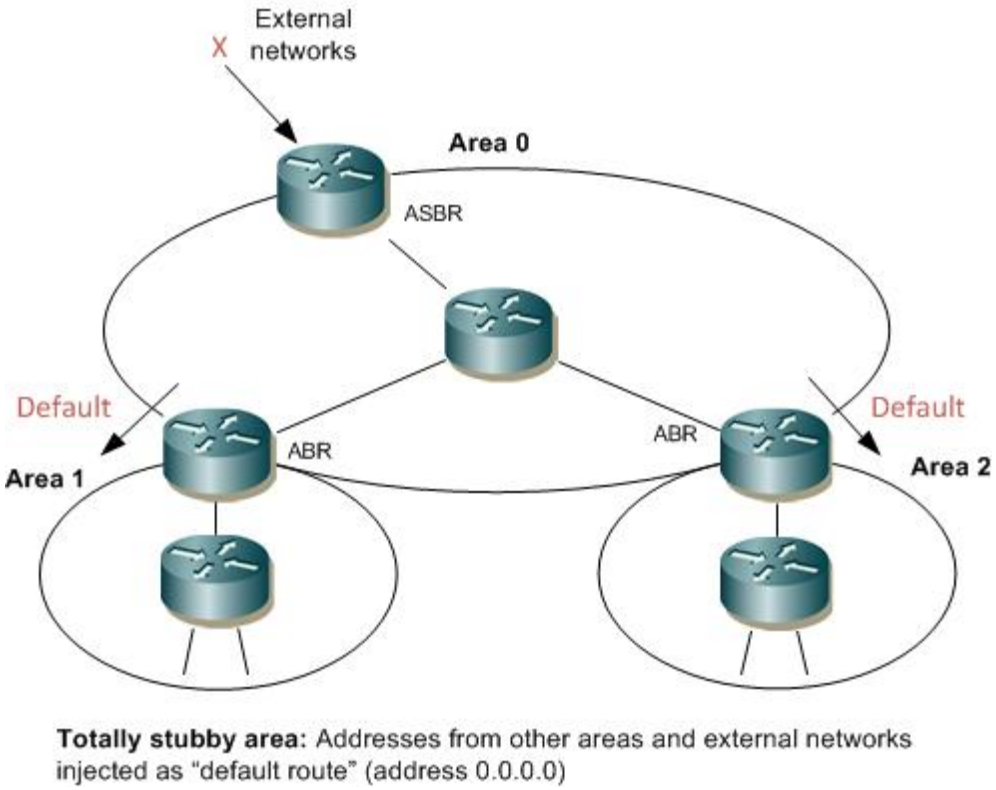


Figure 1-4: Totally Stubby Area

Not So Stubby Area

Not-so-stubby areas (NSSAs) as shown in Figure 1-5 are an extension of stub areas. Like stub areas, they suppress routing advertisements in the area, relying instead on default routing to external destinations. As a result, NSSAs (like stub areas) are placed at the edge of an OSPF autonomous system.

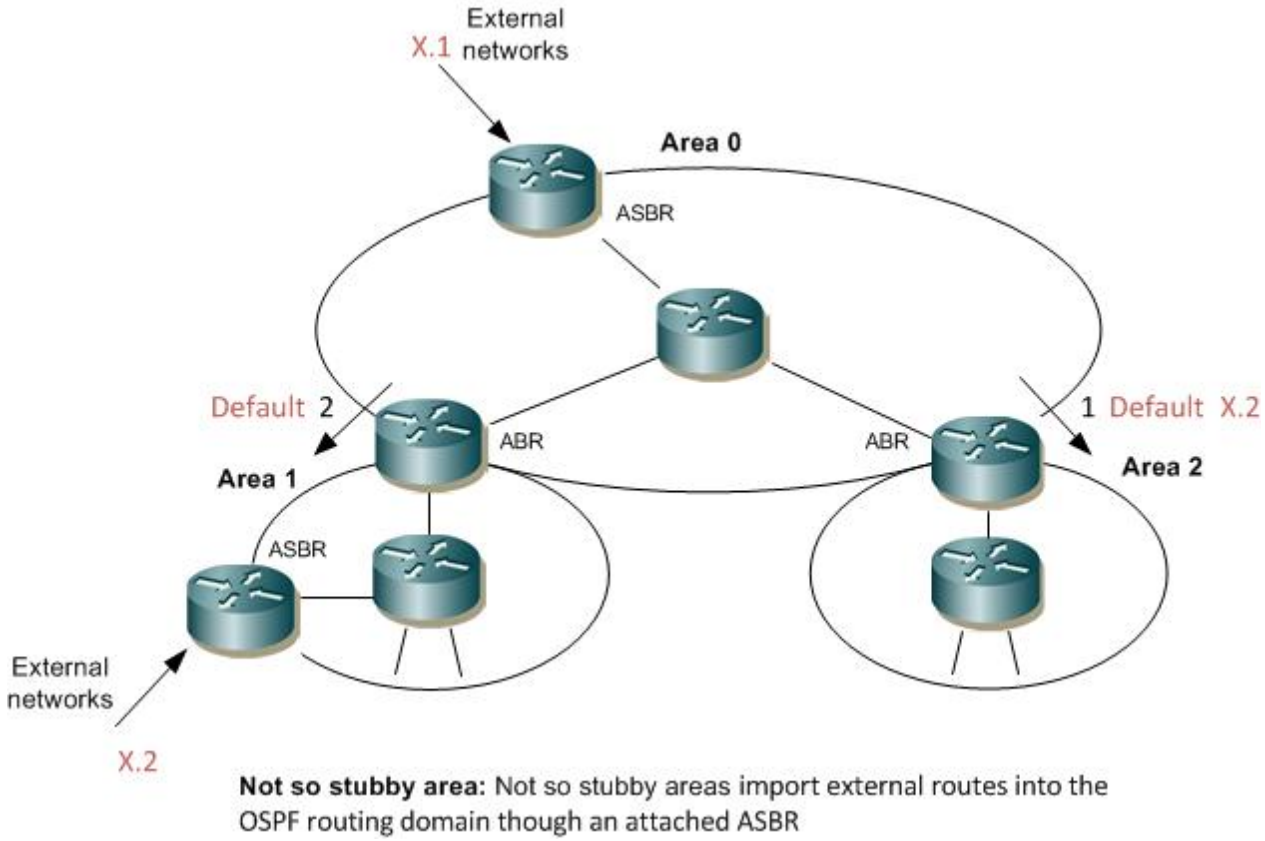


Figure 1-5: No So Stubby Area (NSSA)

However, an NSSA is more flexible than a stub area because an NSSA has an attached ASBR that imports external routes into the OSPF autonomous system, providing transit service to network domains that are not part of the OSPF autonomous system.

The link state databases maintained within the NSSA contain the default route, intra-area and inter-area routes, and the external routes generated by the ASBR in the NSSA.

Area Types Summary

Table 1-2 summarizes the types of OSPF areas.

Table 1-2: Area Types Summary

Area	Description	LSA Handling
Backbone and regular	Accepts intra-area, inter-area, and external routes. The backbone area is the central area to which all other areas in OSPF connect.	None
Stub	Accepts inter-area and intra-area routes, and the default route, but does not accept routes to external systems. To reach outside networks, the routers in the area use the default route injected by the ABR.	Type 5 LSAs are replaced with the default route
Totally Stub	Accepts only intra-area routes and the default route. To send traffic outside the area, the routers in the area use the default route injected by the ABR.	Type 3, 4, and 5 LSAs are replaced with the default route
NSSA	A stub area containing an ASBR that accepts external routes.	Type 7 LSAs originate at the NSSA ASBR and convert to Type 5 at the NSSA ABR
NSSA Totally Stub	A totally stub area containing an ASBR that accepts external routes.	Type 3, 4, and 5 LSAs are replaced with the default route Type 7 LSAs originate at the NSSA ASBR and convert to Type 5 at the NSSA ABR

Virtual Links

All areas in an OSPF autonomous system must be physically connected to the backbone area (area 0). In cases where this physical connection is not possible, you can use a virtual link as shown in Figure 1-6 to:

- Connect to the backbone through a non-backbone area
- Connect two parts of a partitioned backbone through a non-backbone area

The area through which you configure the virtual link, known as a transit area, must have full routing information. The transit area cannot be a stub area.

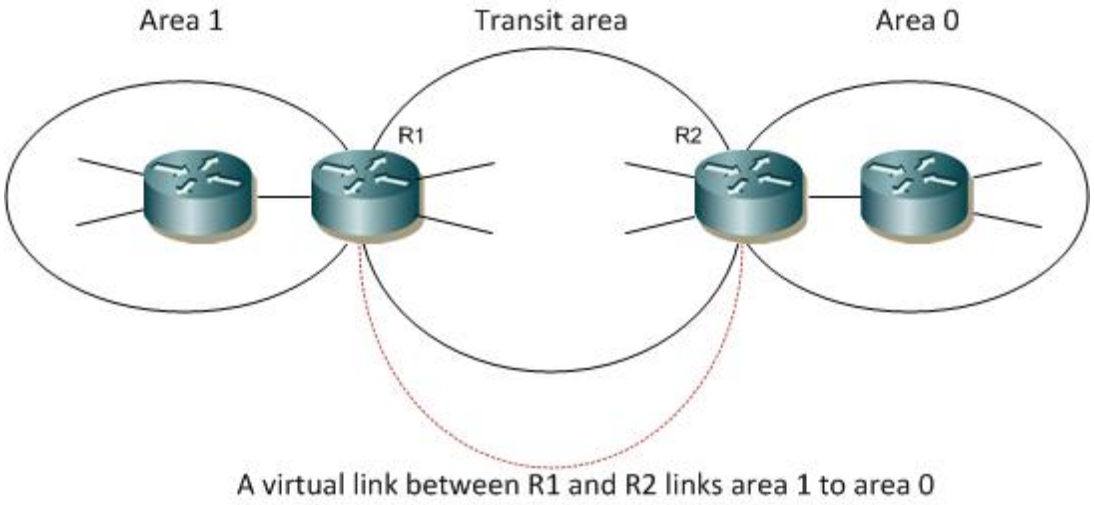


Figure 1-6: OSPF Virtual Link

A virtual link is established between two ABRs sharing a common non-backbone area. The link is treated as a point-to-point link. The common area is known as a transit area.

Virtual links are not recommended and should be a temporary fix since they add complexity to the network topology and are difficult to troubleshoot.

OSPF Routing Example

In Figure 1-7:

- For Host 1 in area A to reach Host 6 in area C (an inter-area transfer), the packets must travel, beginning at R3 through ABR2, R5, and ABR3; then through R7 and R8 to Host 6.
- For Host 5 to reach Host 4, there are two possible paths: one hop through R6 or two hops through R4 and R5. Depending on the sum of the link costs, the one-hop path may be less preferable.

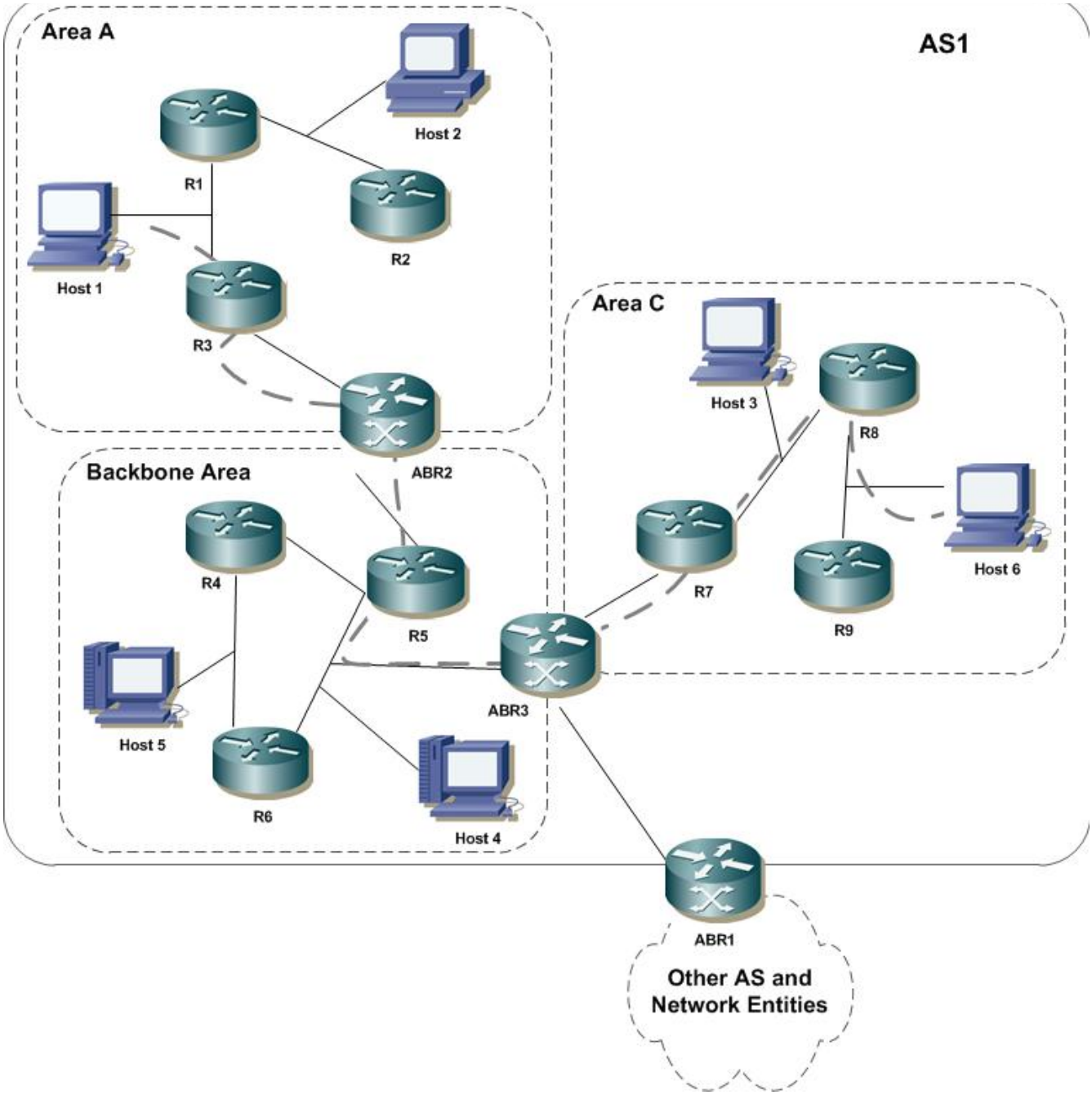


Figure 1-7: OSPF Routing

Link-State Advertising

Flooding is used to distribute and synchronize the link-state database between OSPF routers. In flooding, link-state-advertisements (LSAs) are exchanged between adjacent OSPF routers. The contents of an LSA describe an individual network component (that is, router, link, or external destination).

When a router generates or modifies an LSA, it must communicate this change throughout the network. The router starts this process by forwarding the LSA to each adjacent device. Upon receipt of the LSA, these neighbors store the information in their link state database and communicate the LSA to their neighbors. This store-and-forward activity continues until all devices receive the update. This process is called reliable flooding. Two steps are taken to ensure that this flooding effectively transmits changes without overloading the network with excessive quantities of LSA traffic:

- Each router stores the LSA for a period of time before propagating the information to its neighbors. If, during that time, a new copy of the LSA arrives, the router replaces the stored version. However, if the new copy is outdated, it is discarded.
- To ensure reliability, each link state advertisement must be acknowledged. Multiple acknowledgements can be grouped together into a single acknowledgement packet. If an acknowledgement is not received, the original link state update packet is retransmitted.

Link state update packets are flooded throughout the OSPF autonomous system based on their type:

- The flooding procedure starts when a link state update packet has been received
- Each LSA must be acknowledged separately

When a router encounters two instances of an LSA, it must determine which is more recent. The LS sequence number, LS age, and LS checksum fields are used to determine which instance is more recent.

Opaque LSAs

ZebOS-XP supports OSPF Opaque LSA based on RFC 5250. Opaque LSAs, a class of LSAs, provide a mechanism to extend OSPF functionality.

Opaque LSAs consist of a standard LSA header followed by a 32-bit field used for application-specific information. Standard OSPF link-state database flooding mechanisms are used to distribute Opaque LSAs to all or some limited portion of the OSPF topology. The information field may be used directly by OSPF or indirectly by some application to distribute the information throughout the OSPF domain.

The following are the three Opaque LSA types (described in the Table 1-3):

- LSA type 9
- LSA type 10
- LSA type 11

Link-State Advertisement Summary

Table 1-3 summarizes the LSA types.

Table 1-3: LSA Summary

LSA Type	IPv4 name	IPv6 name	Generated by	Advertises	Where flooded
Type 1	Router LSA		All routers in an area	The routers' directly attached interfaces (intra-area routes); also indicates if the router is ABR or ASBR	Each area to which the router belongs
Type 2	Network LSA		The designated router of a segment	The neighbors connected to the link; these do not leave the area	Each area to which the DR belongs
Type 3	Summary LSA	Inter-Area-Prefix LSA	ABR	A route to neighbors outside the area (inter-area routes); ABRs originate a single summary-LSA for each known inter-area destination	Only intra-area routes are advertised into area 0; both intra-area and inter-area routes are advertised into all other regular areas; not flooded into totally stub areas
Type 4	ASBR Summary LSA	Inter-Area-Router LSA	ABR	A route to an ASBR	Backbone area which then sends the LSA to other areas; not flooded into stub areas
Type 5	External LSA	AS-External LSA	ASBR	A redistributed external route	Entire autonomous system
Type 6	Deprecated		N/A	N/A	N/A
Type 7	NSSA External LSA		ASBR inside an NSSA	A route redistributed into the NSSA; Type 7 is translated into Type 5 by the ABR as it leaves the NSSA.	NSSA area only
Type 8	N/A	Link LSA	All routers	Link-local addresses and a list of IPv6 addresses on the link	Link local only
Type 9	Link-local Opaque LSA	Intra-Area-Prefix LSA	All routers	IPv4: Application-specific information IPv6: Prefixes for stub and transit networks	IPv4: Link local only IPv6: Area only
Type 10	Area-local Opaque LSA	N/A		Application-specific information	Area only
Type 11	AS Opaque LSA	N/A		Application-specific information	Entire autonomous system

Note: OSPFv3 does not provide opaque LSA types. Instead, any unknown LSA type is always accepted into the local copy of the LSDB and flooded according to the flooding scope encoded in the LSA.

Link-State Advertisement Header

All OSPF LSAs have a 20 byte common header as shown inTable 1-4.

Table 1-4: LSA Header

Field	Description
LS Age	The time in seconds since the LSA was originated.
Options	Optional capabilities.
LS Type	Function performed by the LSA: Hello: This packet type discovers and maintains neighbor relationships. Database description: This packet type describes the set of LSAs contained in the router's link state database. Link state request: This packet type requests a more current instance of an LSA from a neighbor. Link state update: This packet type provides a more current instance of an LSA to a neighbor. Link state acknowledgement: This packet type acknowledges receipt of a newly received LSA.
Link State ID	The IP address of the link
Advertising Router	The identifier of the router originating the LSA
LS Sequence Number	Assigned to each LSA
LS Checksum	Checksum for the LSA
Length	The length of the LSA

Hello Protocol

The Hello Protocol is responsible for establishing and maintaining neighbor relationships:

- It ensures that communication between neighbors is bi-directional. Bi-directional communication is indicated when the router sees itself listed in the neighbor's Hello Packet.
- On broadcast and NBMA networks, the hello protocol elects a Designated Router for the network.
- Hello packets are sent out each functioning router interface.
- The hello packet indicates how often a neighbor must be heard from to remain active (RouterDeadInterval) and the interval between Hello Packets sent out (HelloInterval)
- Both HelloInterval and RouterDeadInterval must be the same for all routers attached to a common network

Designated Routers

Each multi-access (Ethernet) network elects a designated router (DR) and backup designated router (BDR). The DR performs two key functions on the network segment:

- It forms adjacencies with all routers on the multi-access network. This causes the DR to become the focal point for forwarding link-state advertisements.
- It generates network link advertisements listing each router connected to the multi-access network.

The BDR forms the same adjacencies as the designated router. It assumes DR functionality if the DR fails.

Network Types

OSPF categorizes network segments into these types:

- Point-to-point networks directly connect a single pair of routers.
- Multi-access networks are capable of connecting more than two routers. They are further subdivided into two types:
  - Broadcast multi-access networks have the capability of simultaneously sending a packet to all attached routers. Ethernet is an example of an OSPF broadcast multi-access network. OSPF routers on a broadcast network elect a designated router and a backup designated router.
  - Non-broadcast networks are capable of connecting more than two routers but have no broadcast capability. Each packet must be specifically addressed to every router in the network.
    - Neighboring routers are maintained on these nets using OSPF's Hello Protocol. However special configuration is required to discover these neighbors.
    - OSPF protocol packets that are normally multicast need to be sent to each neighboring router.
    - OSPF runs over non-broadcast in two modes: NBMA and point-to-multipoint.
      - Non-Broadcast Multiple Access (NBMA) mode emulates a broadcast network. Frame Relay, X.25, and ATM are examples of NBMA networks.
      - Point-to-multipoint networks are a special case of NBMA networks where a router is not required to have a direct connection to every other device. All router-to-router connections are treated as point-to-point connections and there is no DR and no network LSA generated. This is known as a partially meshed environment.

Route Summarization

Route summarization is the process of consolidating multiple contiguous routing entries into a single advertisement. This reduces the size of the link state database and the IP routing table. In an OSPF network, summarization is performed at an area border router. There are two types of summarization:

- Inter-area route summarization is performed by the ABR for an area. It is used to summarize route advertisements originating within that area. The summarized route is announced into the backbone. The backbone receives the aggregated route and announces the summary into other areas.
- External route summarization applies specifically to external routes injected into OSPF. This is performed by the ASBR that is distributing the routes into the OSPF network.

OSPF Authentication

There are three types of OSPF authentication:

- Null (Type 0): routing exchanges over the network are not authenticated.
- Simple text (Type 1) uses clear-text passwords. You set the same password for all neighbors that communicate in a network.
- Message Digest Authentication (MD5) (Type 2) is cryptographic authentication. You configure a key and a key identifier for all neighbors that communicate in a network. The router generates a message digest on the basis of the key, key ID, and OSPF packet, and adds it to the OSPF packet.

You can configure he authentication type on a per-interface basis or a per-area basis. Additionally, interface and area authentication can be used together. If the interface authentication type is different than the area authentication type, the interface authentication type overrides the area authentication type. If the authentication type is not specified for an interface, the authentication type for the area is used.

ZebOS-XP Architecture

The following section provides a brief overview of the internal architecture of the OSPF module and the external interfaces to other components in ZebOS-XP.

Architecture Overview

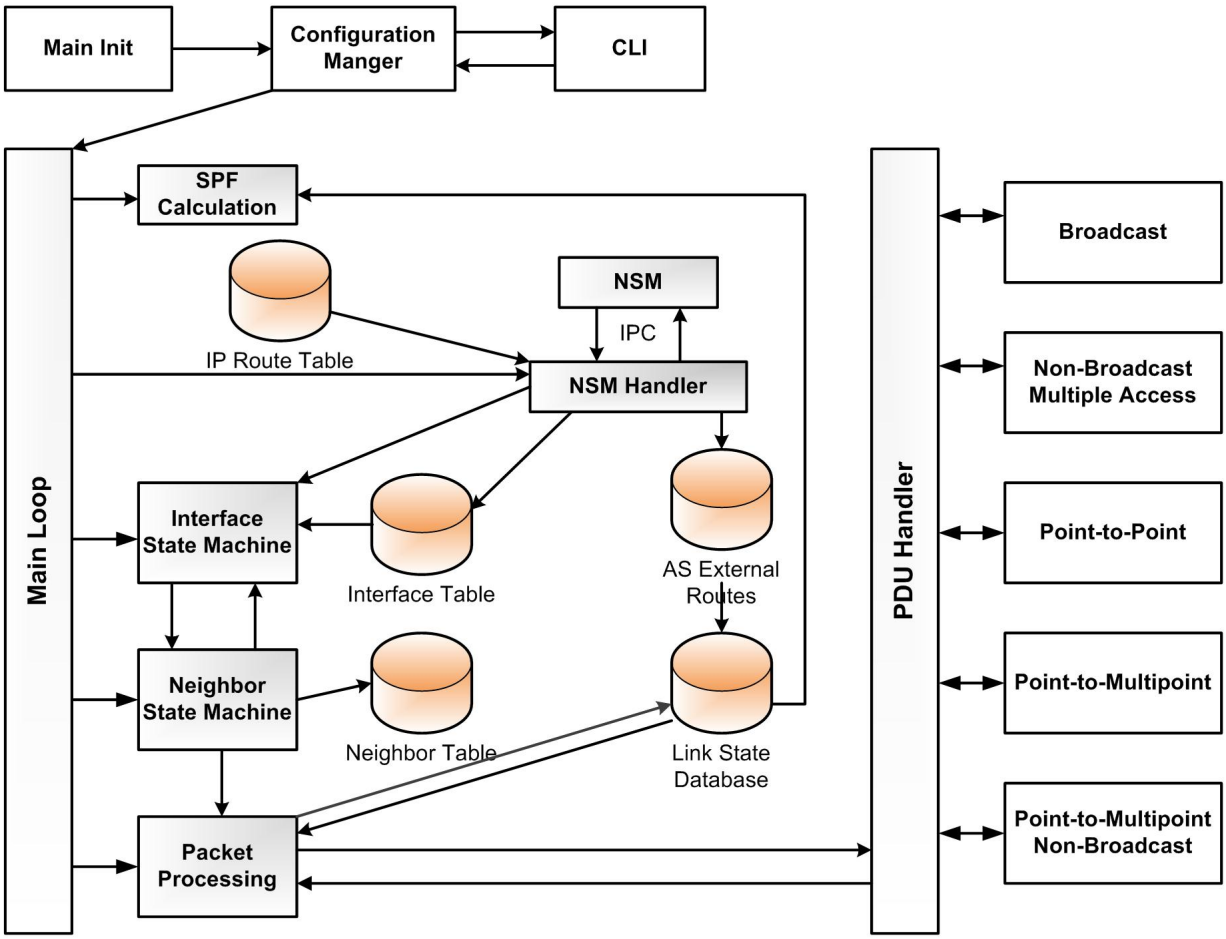


Figure 1-8: OSPF Architecture

The internal architecture of the OSPF module is shown in Figure 1-8 above. The OSPF module runs as a single task or process in one main loop. This main loop schedules sub-tasks for each component described below.

PDU Handler. The PDU handler is the entry point for all OSPF control packets and is the interface between the TCP/IP. stack and the OSPF module.

Packet Processing. The packet processing task handles the encoding/decoding of control packets and initiates the . Link State Database Update where applicable.

Link State Database. The link state database (LSDB) holds the details of all Link State Advertisements received from. neighbors. An update to link state database triggers an SPF calculation.

SPF Calculation. This component handles the SPF calculation triggered by an LSDB update. The output of the SPF . calculation is the SPF tree followed by route installation in the OSPF Route Table.

OSPF Route Table. The OSPF route table holds the selected route with next hop details and is derived from the SPF calculation process.

NSM Message Handler. This is the NSM client residing within the OSPF process, which handles Inter Process Communication between NSM and OSPF. NSM shares the following information with OSPF:

- Layer 3 Interface Information
- Redistributed Routes (Static, RIP, BGP)

Interface Table. OSPF holds a local copy of the interface information shared by NSM.

AS External Routes. The AS External route table is populated from routes redistributed by NSM.

Interface State Machine. The interface state machine handles state machine events for all interface related updates received from NSM

Neighbor State Machine. The neighbor state machine handles all state machine events for protocol state information received from neighbors.

Neighbor Table. The neighbor table maintains the list of neighbors for each logical OSPF interface in the system.

Configuration Manager. The configuration manager manages all protocol configuration through CLI and SNMP.

Architecture

Figure 1-9 below illustrates the overview of OSPF module interaction with NSM, IMI and TCP/IP Stack.

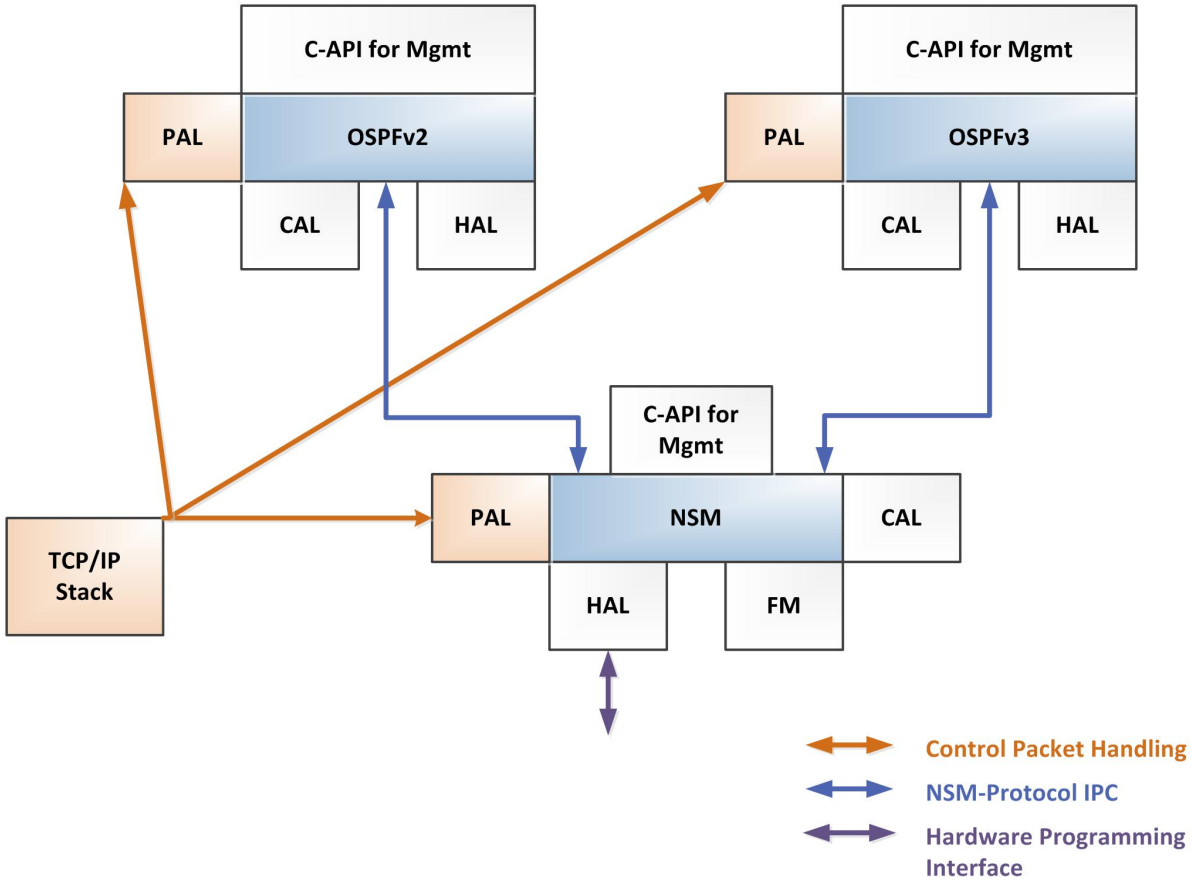


Figure 1-9: OSPF Architecture

CHAPTER 2 Graceful Restart

This chapter describes the operation of Graceful Restart for OSPFv2 and OSPFv3.

OSPFv2 Graceful Restart

The ZebOS-XP OSPF graceful restart feature is based on `draft-ietf-ospf-hitless-restart-08`. Routers that separate control and management tasks from data-forwarding tasks are well-suited to the graceful restart feature. Network personnel initiate graceful restarts. Graceful restart is possible when the network topology is stable, and the restarting router retains its forwarding tables.

Normal Restart

Under normal conditions, OSPF routers automatically route around a restarting router. With graceful restart, a restarting router announces that it is entering a restart state with a Grace linked-state advertisement (LSA). Neighboring routers continue to announce the restarting router as if it were still adjacent. When the router control is back online, the router is back online with no time lost to reconstructing forwarding tables across the topology.

Graceful restarting is particularly suited to planned network outages. It might work in some unplanned outages, but generally the network has too little time to prepare and save the tables in a restartable state.

The layout of a Grace LSA contains several fields:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LS age            |     Options   |       9       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       3       |                     0                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Advertising Router                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      LS sequence number                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         LS checksum           |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                            TLVs                             -+
|                            ...                                |
```

Restarting Mode

A router that sends out, at network personnel initiation, a Grace LSA is termed a restarting router. The interval of time from when the Grace LSA is sent until all the adjacencies are fully re-established is called the graceful restart. During graceful restart, the restarting router:

- does not originate any LSAs of type 1-5 or 7. The intent is for adjacent routers to calculate routes using the LSAs sent prior to the restart.

- does not install OSPF routes into the forwarding tables, relying on the forwarding entries extant prior to restart.
- retains its Designated Router (DR) status if it finds an entry in a Hello packet listing the restarting router as the DR.

Entering Restart Mode

A router enters restart mode when network personnel issue the `restart ospf graceful` command. The restarting router uses the `nsm_client_send_preserve_with_val` function to cause NSM to preserve the forwarding table. The restarting router also preserves, in non-volatile memory, the cryptographic sequence number using TLV fields: the `nsm_client_send_preserve_with_val` function accomplishes this.

If MD5 authentication is enabled, the cryptographic sequence number is included.

When the forwarding table and cryptographic data are preserved, Grace LSAs are sent, one for each of the OSPF interfaces. Graceful restart then proceeds with reloading/restarting the router.

Exiting Restart Mode

Graceful restart is exited when:

- All adjacencies are re-established.
- The restarting router receives an LSA that is not consistent with its pre-restart LSAs.
- The grace period expires.

At exit time, the router resumes normal routing duties, regardless of restart success. The `ospf_hitless_restart_exit` procedure accomplishes these actions:

- re-originates its LSAs for all attached areas.
- re-originates network LSAs for segments it is the DR
- recalculates the routes, installing results into the system forwarding table, originating summary LSAs, Type-7 LSAs and AS external LSAs, each as necessary.
- removes remnant entries from the system forwarding table
- flushes self-originated Grace LSAs

Helper Mode

A router that receives and accepts a Grace LSA is called a helper router. This router continues to advertise the restarting router. It sets the OSPF_NEIGHBOR_GRACEFUL flag in the `struct ospf.neighbor`. When this router exits from helper mode, it resets this flag.

Entering Helper Mode

Entrance into helper mode is contingent upon several conditions, not the least of which is local policy.

- Helper router has full adjacency with the restarting router.
- There are no changes to the link-state database since the initiation of the restart.
- The grace period (given in the Grace LSA) has not expired.
- Local policy is followed. (Examples of prohibitive policies are: never act as a helper; only at certain times of day; and only for certain routers.)

Note: Any router can act as a helper for multiple restarting routers. Grace periods can be updated if subsequent Grace LSAs are received.

Exiting Helper Mode

A helping router remains in helper mode until one of the following occurs:

- Restart is successful. When the Grace LSA is flushed, this is the signal that the restart was successful. The helper router calls `ospf_restart_helper_exit_by_lsa ()` (located in `ospf_restart.c`)
- The grace period expires. When the helping router receives a MaxAge Grace LSA, it terminates the helper mode calling `ospf_restart_helper_exit_timer ()` (located in `ospf_restart.c`).
- A change in the link-state database. When a helping router installs a new LSA in its database (Types 1-5, 7) with both:
  - The contents of the LSA changed (not a mere refresh)
  - The LSA would have been flooded to the restarting router if they would have been fully adjacent.

Note: If multiple helper routers are involved, any one of them can terminate the restart.

Source Code

The code for ZebOS-XP OSPF graceful restart can be found in the `ospf_restart.c`, `ospf_restart.h`, and `ospf_nsm.c` files. The `ospf_nsm.c` file adds graceful-restart support to the NSM.

ospf_restart.h

This file contains constants and structures for graceful restart.

```
OSPF_GRACE_PERIOD_DEFAULT                    60
OSPF_GRACE_PERIOD_MIN                         1
OSPF_GRACE_PERIOD_MAX                      1800
OSPF_RESTART_STALE_TIMER_DELAY               40
OSPF_RESTART_STATE_TIMER_INTERVAL             3
OSPF_GRACE_LSA_TLV_DATA_SIZE_MAX             24
OSPF_GRACE_LSA_GRACE_PERIOD                   1
OSPF_GRACE_LSA_RESTART_REASON                 2
OSPF_GRACE_LSA_IP_INTERFACE_ADDRESS           3
OSPF_GRACE_LSA_GRACE_PERIOD_LEN               4
OSPF_GRACE_LSA_RESTART_REASON_LE              1
OSPF_GRACE_LSA_IP_INTERFACE_ADDRESS_LEN       4
```

ospf_restart.c

This file contains the procedural code for the graceful restart.

ospf_api.c

This file contains the functions that perform the CLI commands.

ospf_cli.c

Contains the `defun` statements for the CLI commands. See the *Open Shortest Path First Command Reference* for details.

ospf_nsm.c

Contains support code for graceful restart bracketed by:

```
#ifdef HAVE_RESTART ... #endif HAVE_RESTART
```

ospf_nsm.h

This file contains function prototypes for the NSM support of graceful restart.

OSPFv3 Graceful Restart

OSPFv3 graceful restart lets a router gracefully restart the OSPFv3 control plane by maintaining the data-forwarding plane. The data-forwarding plane of a router can continue to process and forward packets, even if the control plane restarts.

After a router restarts/reloads, it changes its OSPFv3 processing, until it re-establishes full adjacencies with all of its former fully adjacent neighbors. Graceful restart is the time period between the restart/reload and re-establishment of adjacencies.

Features

ZebOS-XP supports the OSPFv3 graceful restart features listed below.

- `restart ipv6 ospf graceful` CLI.
- CLI to enable helper policies on graceful restart helper.
- Preserving the restart TLVs like grace-period, restart reason, and link state IDs of router, network, external, inter-area, and link LSAs.
- Retrieving the restart TLVs from NSM.
- Framing and origination of the Grace LSA.
- Changes in OSPFv3 processing during the graceful restart period, as specified in RFC 3623 section 2.
- Actions by the restarting router before entering graceful restart, as described in RFC 3623 section 2.1.
- Exit conditions from graceful restart, as described in RFC 3623 section 2.2.
- Capability to revert back to normal OSPFv3 operation on exit from graceful restart, as described RFC 3623 section 2.3.
- Helper neighbor graceful restart.
- Actions by helper neighbor while entering and exiting the helper mode, as described in RFC 3623 section 3.

System Architecture

The following figure illustrates and describes graceful restart functionality between an OSPFv3 restarting router and NSM.

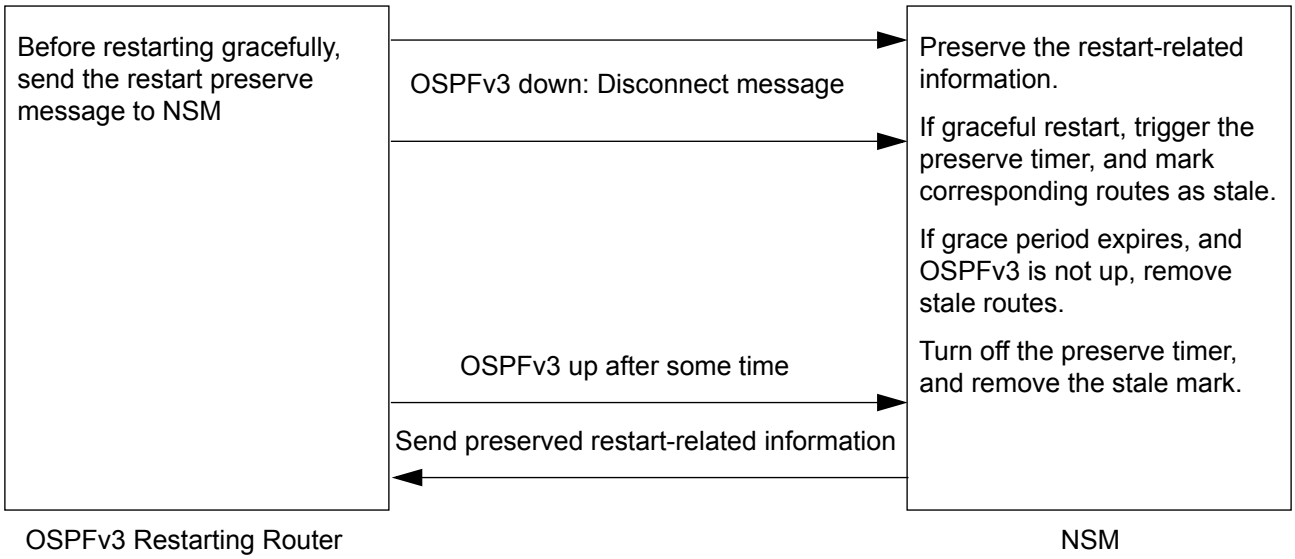


Figure 2-1: Graceful Restart Functionality

Whenever the `restart ipv6 ospf graceful` command is given, the restarting router ensures that its forwarding table(s) are updated before entering graceful restart mode. The ospf6d module shuts down after performing the following actions:

- Preserve a grace period and restart reason by sending a message to NSM.
- Notify NSM it is going to be restarted gracefully.
- Frame a Grace LSA by keeping the TLVs related to the grace period and restart reason. In this case, the Grace LSA is Type 0x000b.
- Wait for finite time (2 seconds) for acknowledgment of the Grace LSA. If not received, it retransmits the Grace LSA until acknowledged. The acknowledgement wait time is a static value (2 seconds).

When the daemon is up, it receives the preserved data from NSM, then schedules the restart timer for the remaining grace period.

While sending hellos, the graceful restarting state is monitored (completed or not): the adjacency state of the restarting router, with all of its neighbors, is checked.

- If the restarting router has reached full state adjacency with all of its neighbors, graceful restart mode is exited: successful restart.
- While in restarting mode, if an inconsistent router LSA has been received from any of its neighbors, graceful restart mode is immediately exited: unsuccessful restart.

If the grace period expires, restart mode is exited. Upon exiting,

- The router LSA for all areas is re-originated.
- If a DR, the network LSA is re-originated.
- The invalid self-originated Type-5 and Type-7 LSAs are flushed.
- The self-originated Type-5 and Type-7 LSAs are updated.
- The Grace LSA is flushed.
- The SPF calculation is re-run.

Helper Functionality

Whenever a restart-capable router receives a Grace LSA, helper policies are applied. The existing helper policies are:

- Never: Never act as restart helper
- Only-reload: Help only on software reloads
- Only-upgrade: Help only on software upgrades
- Max-grace-period: Help only if the received grace-period is less than this value

If the Grace LSA meets all specified helper policies, that router enters into helper mode. If the router is not restart capable, the received Unknown-type LSA is treated as a link-local-scoped LSA.

Exiting conditions from helper mode include:

- The Grace LSA originated by the restarting router on the segment is flushed.
- The grace period expired.
- A change in the link-state database contents indicates a network topology change, which forces graceful-restart termination.

Exiting actions include:

- Re-calculation of the DR for the segment.
- Re-origination of its router LSA for the segment's OSPF area.
- If the DR for the segment, it re-originates the network LSA for the segment.
- If the segment was a virtual link, its router LSA for the virtual link's transit area is re-originated.

Data Flow

The following illustrates the data flow of communication with NSM and neighbors.

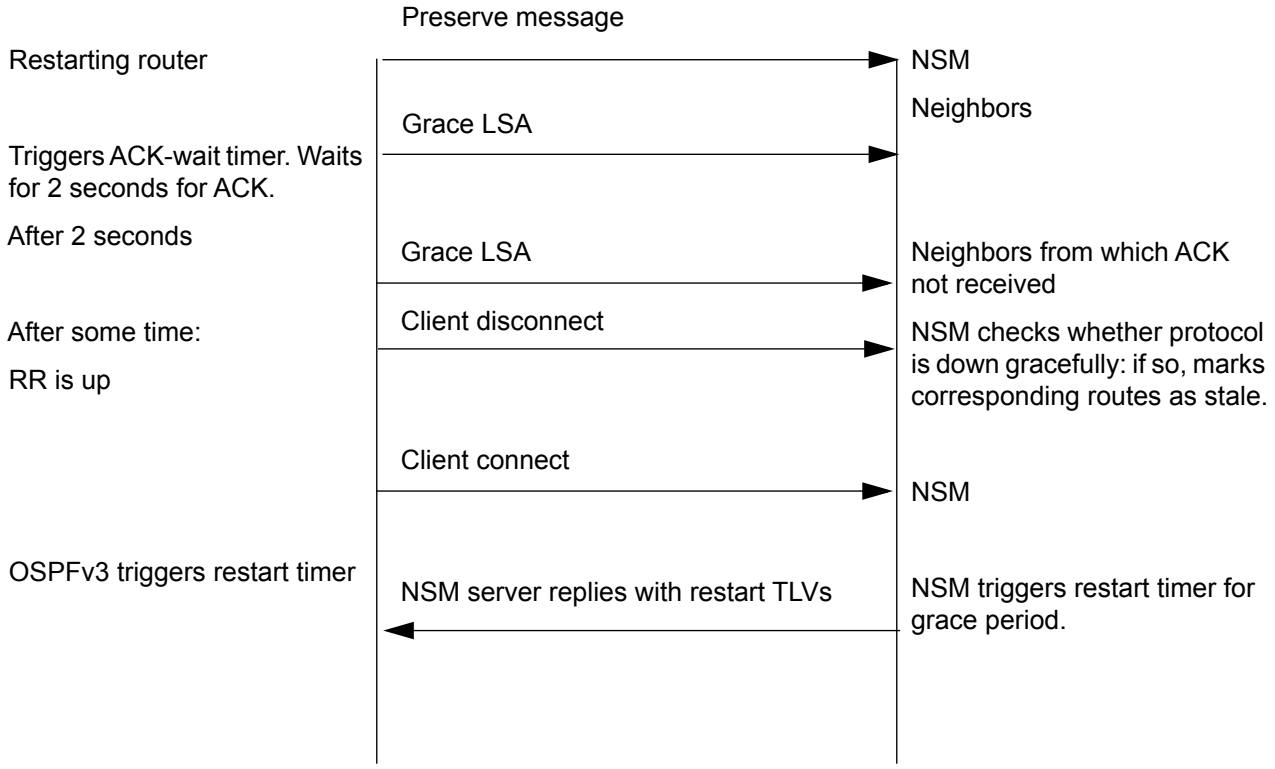


Figure 2-2: OSPF Graceful Restart Data Flow

Command API

For information about the OSPFv2 graceful restart API functions, refer to OSPFv2 Graceful Restart API.

For information about the OSPFv3 graceful restart API functions, refer to OSPFv3 API.

CHAPTER 3 Optimization

ZebOS-XP OSPF optimization includes support for the following features:

- SPF calculation optimization
- Congestion detection and avoidance
- MaxAge Walker optimization
- Controlled incremental SPF calculation
- SPF exponential hold-time backoff

These features are described in the following sections.

SPF Calculation Optimization

Shortest Path First (SPF) calculation optimization reduces total SPF calculation time to enhance scalability and provide faster convergence.

SPF calculation time in OSPF is optimized by implementing the candidate list using a binary-heap data structure, instead of a linked-list data structure. The binary-heap data structure is in the ZebOS-XP common library module. This data structure is used compactly for implementing the candidate list in the SPF calculation to significantly improve the SPF computation in OSPF, particularly for large networks.

SPF calculation in ZebOS-XP is based on the Dijkstra algorithm. By using a binary-heap structure to store the candidate list, the amortized cost for all SPF calculation operations are reduced, resulting in the optimal implementation of the Dijkstra algorithm.

Another advantage of the binary-heap data structure is that it can be stored compactly: No space is required for pointers; instead, the parent and children of each node can be found by simple arithmetic on array indices.

Congestion Detection and Avoidance

When network congestion occurs, a major source of congestion is Link State Advertisement (LSA) retransmissions. ZebOS-XP solves this problem by controlling LSA generation and retransmissions, thus, controlling LSA flooding when an OSPF network becomes congested. This solution improves the scalability and stability of large OSPF networks.

LSA Retransmission Interval

Congestion occurs when the percentage of unacknowledged LSAs to a neighbor is higher than 75 percent. If congestion is detected for a neighbor, the retransmission interval for all LSAs to the neighbor is exponentially increased using the following exponential back-off algorithm:

R(1) = Rmin

R(i+1) = Min(KR(i), Rmax) for i >=1

where:

R(i) represents the LSA retransmission (Rxmt) interval value used during the i-th retransmission of an LSA.

K, Rmin, and Rmax are constants

Min(.,.) represents the minimum value of its two arguments

The default Rxmt interval is always taken as Rmin. The default Rxmt interval is 5 seconds, if not configured using the retransmit command.

Rmax is 8 times the value of Rmin. If the Rmin value is changed using the OSPF retransmit command, the Rmax value will also be changed, however, the Rmax value cannot exceed 65535. For OSPFv2, the command is ip ospf (A.B.C.D|) retransmit-interval <1-65535>. For OSPFv3, the command is ipv6 ospf retransmit-interval <1-65535>.

If the percentage of unacknowledged LSAs drops below 75 percent, but is more than 25 percent, the same retransmission interval is used as the interval used during congestion. The retransmission interval is reduced to the default value (Rmin) only when the percentage of unacknowledged LSAs drop below 25 percent.

Congestion Detection

Congestion is detected at the neighbor, based on the number of unacknowledged LSAs using the following algorithm:

nbr->rxmt_count = x;

nbr->ack_count = y;

then unack_count = x – y;

if (unack_count == 75% of x)

then nbr has congestion;

where:

nbr = neighbor

rxmt = LSA retransmission

Retransmission Lists

ZebOS-XP maintains multiple retransmission lists for each neighbor. LSAs are added to the retransmission list, depending on the value: (current time)%(number of retransmission list). This ensures fair distribution of LSAs to multiple retransmission lists.

The retransmission interval is equally divided among the retransmission lists. For example, if the retransmission interval is 5 seconds, and 5 retransmission lists are used, LSAs from one retransmission list are flooded every second.

If congestion occurs, the LSA retransmission interval is increased, and the retransmission interval is correspondingly increased. For example, if the retransmission interval is increased to 20 seconds, LSAs from every retransmission list are flooded every 4 seconds.

MaxAge Walker Optimization

The OPSF MaxAge Walker Optimization process increases processing efficiency by distributing the CPU load and allowing CPU time for other processes. This optimization avoids unnecessary scans of the entire Link State Database (LSDB), and provides enhanced processing efficiency and improved scalability.

The OSPF_MaxAge_Walker routine was originally called every 10 seconds, scanned through the entire LSDB, and determined which LSAs Max-Aged: This was a very CPU intensive process, especially with LSAs in the order of several hundreds of thousands. Also, frequent scans of the LSDB affected other CPU processes. To optimize the processing efficiency, two solutions are provided:

1. Reduce the number of LSDB scans: Instead of scanning the LSDB every 10 seconds, ZebOS-XP tracks the maximum age of all LSAs in the LSDB. For example, if the maximum age is 30 minutes, the LSDB is not scanned for another 30 minutes. Also, during each LSA addition, the maximum age value is checked and updated.
2. Scan a limited number of LSAs in a single scan: Instead of scanning the entire LSDB, scan in multiple steps. After scanning a certain number of LSAs, the event is suspended, and the next scan event is scheduled. This allows CPU time for other processes.

Controlled Incremental SPF Calculation

ZebOS-XP intelligently determines when not to do the incremental SPF calculation to improve CPU utilization of SPF, thus providing enhanced scalability when a router is heavily loaded under transient conditions.

This is accomplished by controlling SPF calculation for Summary, AS-External, or NSSA LSA types, if there is a high rate of LSAs when a topology change occurs.

SPF Calculation Functionality

OSPF uses a link state in each individual area that makes up the hierarchy and calculates the shortest path tree inside each area using the LSDB.

The LSDB is constructed from information from other routers gathered using LSAs. The SPF algorithm gleans information from these LSAs to prepare the SPF tree. The SPF algorithm is calculated every time there is a change in the topology within an area.

Installing a new LSA in the database, either as the result of flooding or a newly self-originated LSA, may cause the OSPF routing table to be recalculated.

The SPF algorithm is a very CPU intensive process: frequent SPF calculation for each change can affect the other processes running on the same CPU. To prevent frequent SPF calculation after a change in topology is received via the Router or Network LSA, the system waits for a default period of time, called the hold time, before calculating the SPF.

After the SPF tree is calculated for the area, routes to all inter-area destinations are calculated by examining the summaries of the area border routers. Inter-area routing is analogous to forcing a star configuration on the Autonomous System (AS), with the backbone as the hub and each of the non-backbone areas as spokes.

As the packet exits the IGP domain, the correct ABR to use is chosen in exactly the same way area border routers advertising summary routes are chosen. A similar rule applies to Not-So-Stubby Area (NSSA) routes.

Incremental SPF can be used to calculate summary, external, or NSSA route changes, so that there is no requirement to recalculate the entire SPF tree.

Controlled Calculation Solution

Whenever a topology change summary, external, or NSSA LSA is received, incremental SPF is calculated independently for each LSA, which results in increased CPU utilization under transient conditions. To solve this problem, if the rate of these LSAs is high, and the SPF has been scheduled, the incremental SPF calculation is not done: this reduces CPU utilization. This is possible because the SPF calculation computes all routes from the beginning. If it is known the SPF has been scheduled, route calculation can be deferred for these LSAs.

SPF Exponential Hold-Time Backoff

SPF exponential backoff provides the ability to make the hold time variable and changing, based on the frequency of topology changes. This enables increased scalability for frequent topology changes. It also provides for faster convergence when topology changes occur at a slower rate.

OSPFv2 and OSPFv3 use an exponential back-off algorithm for hold time. In addition, the hold time maximum and minimum periods can be configured using the `timers spf exp` command.

This feature provides variable intelligent delay for SPF, depending on the load, to increase scalability and convergence speed.

Hold Time Calculation Algorithm

The hold time delay calculation uses an exponential mechanism, as follows.

1. The CurrHold value indicates the current hold time to use. At the start of processing, the CurrHold value is initialized to the value of the MinHold.
2. 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 value becomes either the SPF incremental value of its current value, or MaxHold, whichever is less.
3. If the topology change notification is received after the CurrHold time into the SPF incremental value, the CurrHold time becomes the value of the MinHold time.

In these cases, the SPF is delayed by the MinHold time, or CurrHold time since the last SPF calculation, which ever is lesser.

Configuring Hold Times

The `timers spf exp` command allows configuration of maximum and minimum hold times. The timers have a granularity in milliseconds to allow the user to more effectively control the SPF hold time delays. The hold time changes exponentially when a topology change occurs.

The `no timers spf exp` command resets the exponential backoff timers to the default exponential backoff timer values. The default values of the maximum hold (MaxHold) and minimum hold (MinHold) times are 50 seconds and 50 milliseconds, respectively.

LSA Throttling Configuration

The OSPF Link-State Advertisement (LSA) throttling feature provides a mechanism to dynamically slow down link-state advertisement (LSA) updates in OSPF during times of network instability. It also allows faster OSPF convergence by providing LSA rate limiting in milliseconds, when network is stable.

The `timers throttle lsa all` command controls the generation (sending) of LSAs. The first LSA is always generated immediately upon an OSPF topology change, and the next LSA generated is controlled by the minimum start interval. The subsequent LSAs generated for the same LSA are rate-limited until the maximum interval is reached. The “same LSA” is defined as an LSA instance that contains the same LSA ID number, LSA type, and advertising router ID.

The `timers lsa arrival` command controls the minimum interval for accepting the same LSA. If an instance of the same LSA arrives sooner than the interval that is set, the LSA is dropped. It is recommended that the arrival interval be less than or equal to the hold-time interval of the `timers throttle lsa all` command.

CHAPTER 4 Multiple Instances

ZebOS-XP supports multiple OSPFv2 and OSPFv3 instances. Multiple OSPF instances assist in creating administrative separation in a large network to segregate customer traffic and associated settings.

Multiple instances are used to create overlay networks in which separate services are routed only towards routers participating in that service, such as voice. The overlay network isolates routes belonging to one service from another service, by exporting routes, applying tags, and filters routes based on tags.

Each protocol instance contains a routing table, applied routing policies, a routing table group, and interfaces that belong to that instance.

ZebOS-XP Functionality

This section describes how ZebOS-XP handles multiple OSPF instances.

- A separate Routing Information Base (RIB) is created for each protocol instance.
- Because the administrative distance is the same for all OSPF instances, NSM additionally considers the metric of the route for the route preference rule.
- NSM passes the OSPF instance ID while sending redistribution route information to OSPF.
- The redistribution of one OSPF instance is allowed into another OSPF routing table.
- The redistribution of other Layer 3 protocols into a specific OSPF instance is allowed.
- Command support is provided for redistribution between OSPF instances and redistribution to a particular OSPF instance.

Multiple OSPF Instances on a Single Interface

ZebOS-XP supports multiple OSPF instances in a single interface by differentiating packets for the various instances sent and received on the same interface. In the packet header, the authentication type field is divided into an instance ID and authentication type.

System Overview

When an OSPF process is enabled in a subnet, it is associated with an instance ID, in the range of 0 to 255. The instance ID is set through the CLI. Similarly, when multiple OSPF processes are enabled in a single subnet, every process has its corresponding instance ID. By default, the instance ID is zero. Instance IDs are encoded in the respective OSPF packet headers when they are sent out. The 16-bit authentication-type field is divided into an 8-bit instance ID and 8-bit authentication type.

Before bringing up the adjacency, the instance IDs are compared: Received packets with an instance ID not equal to one of the configured OSPF instance IDs on the receiving interface are discarded.

This feature is disabled by default. It is enabled using the `enable ext-ospf-multi-inst` command.

System Architecture

Packet Structure

Packets are differentiated for different instances sent and received on the same interface. Each protocol instance is assigned a separate Instance ID. This instance ID is encoded in the packets associated with the OSPF process. The packet structure is illustrated below.

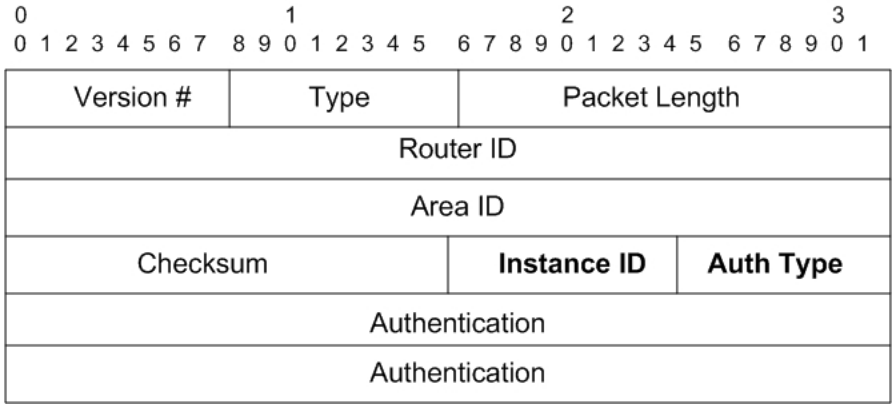


Figure 4-1: Instance ID Packet Structure

Example Configuration

In the following example, OSPF processes 1, 2, and 3 are enabled on a single link with different instances IDs.

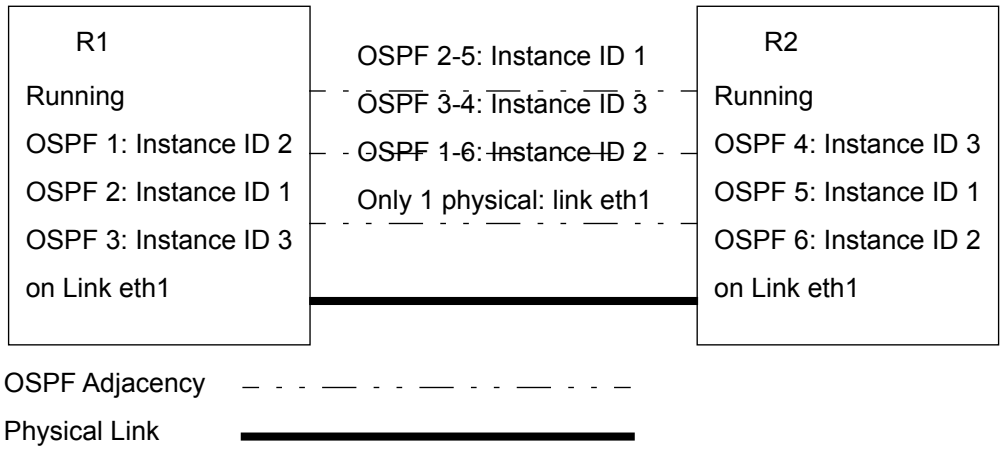


Figure 4-2: Sample Configuration

According to the configuration above, there are the following three OSPF adjacency sessions:

R1		**R2**
OSPF 1	----------	OSPF 6 because instance ID matches
OSPF 2	----------	OSPF 5
OSPF 3	----------	OSPF 4

R1 receives packets from OSPF process 4, 5, and 6 on eth1, but routes packets only from OSPF 6 to OSPF process 1. Similarly, packets from OSPF 5 will be routed only to OSPF 2, and packets from OSPF 4 will be routed only to OSPF 3.

OSPFv3 Address Families

Through the address families feature, OSPFv3 can send and receive both IPv6 and IPv4 unicast traffic. By default, an OSPFv3 process recognizes only IPv6 address prefixes. You can configure an OSPFv3 instance to instead recognize standard IPv4 address prefixes.

While OSPFv3 eliminates some addressing and peering restrictions, it does not support IPv4 nodes as originally defined. To extend OSPFv3 to support IPv4, the OSPFv3 address families feature as defined in RFC 5838 calls for mapping an address family to a separate OSPFv3 instance using the instance identifier. Each OSPFv3 instance maintains its own adjacencies, link state database, and shortest path computation.

RFC 5838 defines the range of instance identifiers as shown in Table 4-1 to use for each address family in OSPFv3.

Table 4-1: OSPFv3 Instance Identifiers

Instance Identifier #	Address Family
0 - 31	IPv6 unicast
64 - 95	IPv4 unicast

Note the following:

- Only one address family can be configured per instance. Multiple router processes can configured per interface, but only one instance per router per interface can be configured. A single IPv4 or IPv6 OSPFv3 process running multiple instances on the same interface is not supported.
- If an OSPFv3 process is IPv4-capable and it receives packets from a process which is not IPv4-capable, then those packets are ignored.

All routers on an OSPFv3 network have an IPv6 forwarding stack. Some (or all) of the routers on the same network can do IPv4 forwarding and be configured with IPv4 addresses.

With the address family feature enabled, the same router can support both IPv4 and IPv6 routing and can advertise IPv4 routes. This capability allows IPv4 routers in different subnets to peer with each other through an IPv6 network, as shown in Figure 4-3.

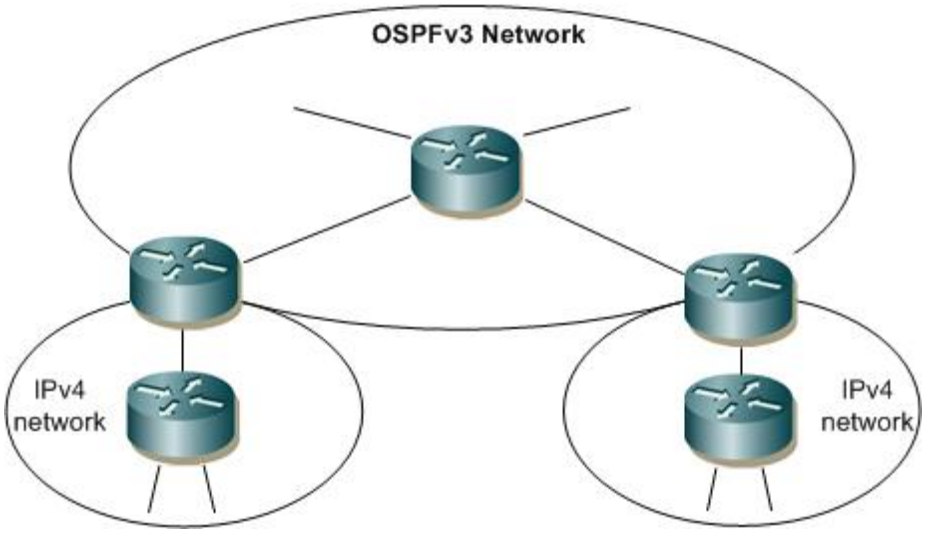


Figure 4-3: Routing IPv4 Subnet Traffic on OSPFv3

As shown ion Figure 4-3, networks running an IPv4 static or dynamic routing protocols exist on the edges of the OSPFv3 network. The OSPFv3 network can forward IPv4 traffic between these networks without tunneling overhead. Any OSPFv3 transit router has both IPv4 and IPv6 forwarding stacks (is dual stack).

As shown in Figure 4-4, the address family feature also allows separate (possibly incongruent) topologies to operate for IPv4 and IPv6 in a single OSPFv3 autonomous system.

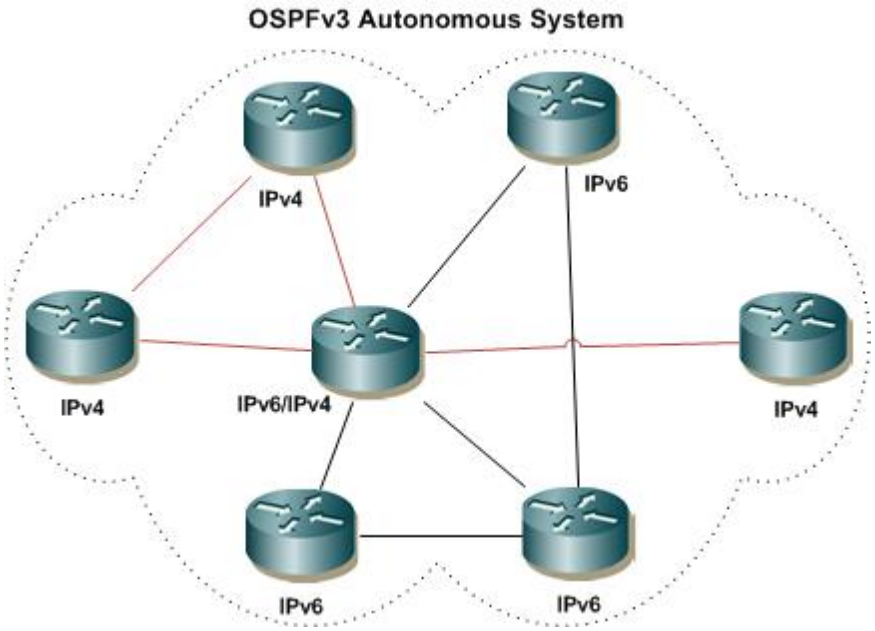


Figure 4-4: Running IPv6 and IPv4 in an OSPFv3 Autonomous System

To use the IPv4 address family in OSPFv3, you must:

1. Configure an IPv4 address on the interface with the `ip address` command
2. Enable IPv6 on the interface with the `ipv6 router ospf` command, selecting an instance identifier in the range for IPv4 addresses (64-95)

You then give the `address-family ipv4 unicast` command to enter address family command mode. After that, you can configure IPv4 unicast routes:

- Summarize intra-area IPv4 routes (`area range` command)
- Summarize IPv4 external routes (`summary-address` command)
- Create a default external route (`default-information originate` command)
- Redistribute IPv4 routes (`redistribute` command)

The IPv4 unicast routes that you configure are separate and distinct form the unicast routes you configure for IPv6.

The IPv4 subnets configured on OSPFv3-enabled interfaces are advertised through intra-area prefix LSAs, just as any IPv6 prefixes. External LSAs are used to advertise IPv4 routes redistributed from any IPv4 routing protocol, including connected and static routes. The IPv4 OSPFv3 process runs the SPF calculations and finds the shortest path to those IPv4 destinations. These computed routes are then inserted in the IPv4 RIB and forwarding occurs natively.

CHAPTER 5 PE-CE for BGP/MPLS VPNs

Using OSPFv2 as the provider-edge/customer-edge (PE-CE) protocol for BGP MPLS VPNs (per RFC 4577) provides easy transition of a service provider's network from the OSPF backbone to the VPN backbone. OSPF routes are redistributed from the remote site of the same area or domain into the CE as Type-3 link-state advertisement (LSA).

Functionality

This feature provides the following:

- CLI support for the OSPF domain identifier and enhancement of the existing CLI for route tagging
- OSPF and BGP communication via NSM as per section 4.1 of RFC 4577
- Sets the DN bit for the Type-3 and Type-5 LSAs sent from the PE to the CE.
- Processes the down (DN) bit for LSAs received from the CE to the PE (RFC 4576)
- Use of the route tag to avoid loops for external routes (Type-5 LSA)
- BGP can process and send the OSPF domain identifier type and OSPF router-type extended community attributes, and can store the attribute, along with VRF routes, when present

System Overview

ZebOS-XP uses a simpler method of Provider Edge (PE) to Customer Edge (CE) router communication in BGP MPLS VPNs than that proposed by VPN-BGP-OSPF. Instead of using Type-3 LSAs, ZebOS-XP propagates VPN routes as Type-5 LSAs, while using multiple instances of OSPF in conjunction with standard BGP/OSPF route redistribution mechanisms. This complies with `draft-ishiguro-ppvpn-pe-ce-ospf-01.txt`.

Because the PE-CE feature requires VRF support, only environments with stacks that support VRF can take advantage of the feature. The environments in which this is acceptable are:

Environment/OS	overlap-vrf	ospf-pece
linux	no	yes
netbsd	no	no
vxworks	no	no
vxworks_ipnet	yes	yes

The ZebOS-XP implementation of the draft is platform independent, requiring VRF support in the stack.

ZebOS-XP BGP VPN is a fully-scalable implementation of IETF 2547-bis. ZebOS-XP BGP can be used either as a PE/P node in the Service provider network, or as CPE to connect to Service provider edge using EBGP. Other ZebOS-XP routing protocols, such as RIP, are also extended to work as CE-PE routing protocols.

ZebOS-XP BGP uses multiple forwarding tables to provide network Isolation of VPN traffic inside the service provider network. A separate default forwarding table can be used to contain public Internet routes and site-based VRF tables for VPN routes. The PE router maintains a VRF table, per site, that shares the same routes. When an IP packet is received on a PE-CE interface, the corresponding VRF is used for the destination IP address lookup.

CE-PE Interaction

The CE devices learn and advertise routes to other sites in its VPNs. ZebOS-XP BGP, OSPF, and RIP can be used for CE-PE route exchange. BGP in PE installs the routes learned in the corresponding VRF table of the site, based the incoming interface.

PE-PE Interaction

ZebOS-XP BGP installs the VPN routes, at the PE device, that were learned from the attached CE, in VRF tables. ZebOS-XP BGP uses Multiprotocol BGP (MP-BGP) to distribute VPN routes to another PE BGP, across the SP network. Routes that are exported to BGP from VRF tables are converted into unique addresses in the VPN-IP address family using Route Distinguishers (RDs). The VPN-IP route is a 12-byte quantity that consists of an RD and the IPv4 address. ZebOS-XP BGP uses route targets to advertise by PE devices with the Router Target attribute. Every VRF is associated with a set of route targets. The import route target list specifies the routes that can be installed in a VRF. Routes that are advertised to other PE devices are tagged with the export route targets.

Traditional OSPF-BGP Routing

Figure 5-1 below shows an example of traditional OSPF-BGP distribution.

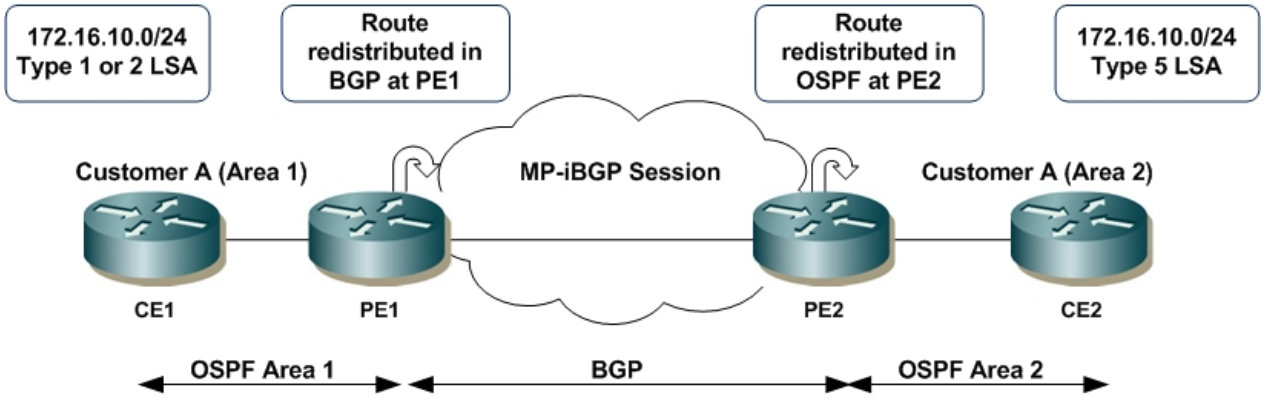


Figure 5-1: Traditional OSPF-BGP Routing

In an MPLS VPN environment, customer networks are connected to an MPLS VPN enabled provider backbone. As shown above. Customer A areas, Areas 1 and 2, are now connected to an MPLS VPN enabled provider network. Areas 1 and 2 have routers CE1 and CE2 running the OSPF routing protocol. MP-iBGP is used between PE1 and PE2 to propagate routes between Site 1 (Area 1) and Site 2 (Area 2). Traditional OSPF-BGP redistribution is performed at PE routers, PE1 and PE2. The figure above depicts the following sequence that occurs in traditional OSPF-BGP redistribution:

1. Network 172.16.10.0/24 is advertised to the PE1 router by CE1 as a Type-1 or Type-2 LSA.
2. Traditional OSPF-BGP route redistribution occurs where 172.16.10.0/24 is redistributed into BGP at PE1. This route is then propagated as a VPNv4 route to PE2.
3. At PE2, the 172.16.10.0/24 BGP VPNv4 prefix is redistributed in OSPF.
4. This redistributed route (172.16.10.0/24) is propagated as an external LSA Type-5 OSPF route.

Therefore, the OSPF route type, or LSA type, is not preserved when the OSPF route for 172.16.10.0 is redistributed into BGP, when traditional OSPF routing rules are used in an MPLS VPN environment. Also, the following characteristics of OSPF external routes do not allow a smooth transition for a customer attempting to migrate from traditional OSPF routing to the MPLS VPN routing model:

- Internal routes, regardless of their cost, are always preferred over external routes

- External routes cannot be summarized automatically
- External routes are flooded throughout all OSPF areas
- External routes could use a different metric type that is not comparable to OSPF cost
- External LSA Type-5 routes are not inserted in stub areas or not-so-stubby areas (NSSAs)

Routing with OSPF as PE-CE Protocol for BGP MPLS VPNs

The following is an example of using OSPF as the provider-edge/customer-edge (PE-CE) protocol for BGP MPLS VPN.

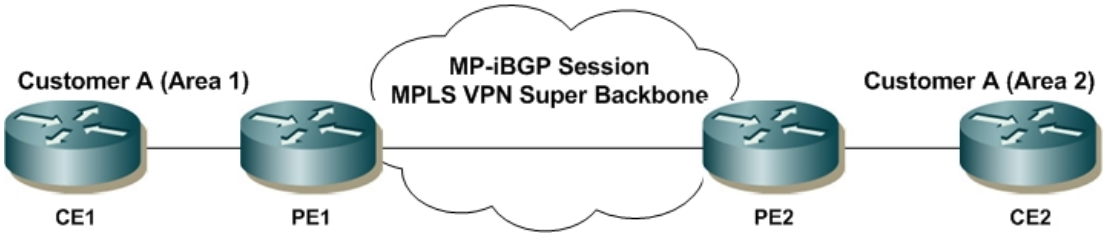


Figure 5-2: OSPF as PE-CE Protocol for BGP MPLS VPNs

To circumvent the issues posed by the traditional OSPF routing model, the MPLS VPN architecture for OSPF PE-CE routing is expanded to allow transparent customer migration from traditional OSPF routing to the MPLS VPN routing model, by introducing another backbone above the OSPF Area 0. This backbone is called the OSPF or MPLS VPN super backbone.

The non-backbone areas, Area 1 and Area 2, are directly connected to the MPLS VPN super backbone that functions as an OSPF Area 0.

The PE routers, PE1 and PE2, which connect OSPF areas in the customer domain to the super backbone, appear as OSPF Area Border Routers (ABRs) for the devices in the customer OSPF domains. CE routers CE1-A and CE2-A are unaware of any other OSPF areas beyond the MPLS VPN super backbone, because it is transparent.

The MPLS VPN super backbone is implemented using MP-iBGP between PE routers. OSPF information is carried across the MPLS VPN super backbone using BGP extended communities. These extended communities are set and used by PE routers.

System Architecture

The domain ID to be assigned to the OSPF instance should be determined so that the route is sent to the other PE router, along with the domain ID value. On the remote PE router, based on the matching domain ID, the route is redistributed as Type-3 LSA (if the domain ID matches) or Type 5-LSA (if the domain ID does not match) to the customer network.

For this purpose, the OSPF domain ID must be assigned for each OSPF instance in the PE router. This information, along with the route type, is carried over to the other PE, and this information is used to identify the exact route type to be sent to the customer network.

The figure below follows the communication between OSPF to NSM and BGP to NSM.

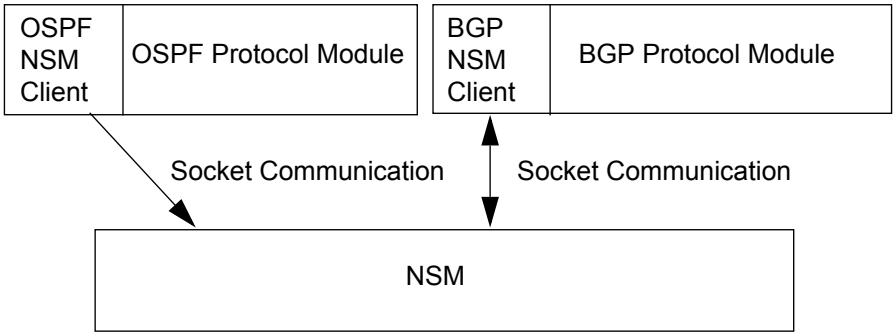


Figure 5-3: Communication Model

The OSPF domain ID is unique for each VRF instance. The OSPF instance is bound to a VRF using the `router ospf vrf` command.

The OSPF module sends a message to the NSM module to update the NSM RIB with a domain ID value.

When the `redistribute ospf` command is given in BGP at the ingress PE, the OSPF route information, along with the domain ID and route-type information, is sent by NSM to BGP, which further sends this VPN route information to the remote PE router. The domain ID value and route type are sent in the form of BGP extended-community attributes to the remote PE. Similarly, when the `redistribute bgp` command is given inside OSPF at the egress PE, the VPN routes received with the matching domain ID are sent with route type received (either Type-3 or Type-5 LSA). Routes with non-matching domain IDs are sent as Type-5 LSA to the customer routers.

Routing loops are prevented in OSPF with the down bit (DN bit) and route tags.

One of the options bit in the LSA header is allocated to be the DN bit. When a Type-3, -5, or -7 LSA is sent from a PE to a CE, the DN bit is set. The DN bit is clear in all other LSA types (per RFC 4576).

The following illustrates the Options field with the DN bit.

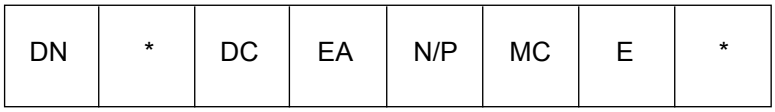


Figure 5-4: Options Field

The DN bit is used between the PE routers to indicate which routes were inserted into the OSPF topology database from the MPLS VPN super backbone, and thus, do not allow redistribution of the route back into the MPLS VPN super backbone. The PE router that redistributes the MP-BGP route as an OSPF route into the OSPF topology database sets the DN bit. Other PE routers use the DN bit to prevent this route from being redistributed back into MP-BGP.

The DN bit stops the routing loops between MP-BGP and OSPF. It cannot, however, stop the routing loops when redistributing the routes between multiple OSPF domains (that is, in the case of external routes). These routing loops are solved using route tags.

The following is an example of using the `redistribute` command with the `tag` value to avoid routing loops in OSPF.

```
# configure terminal
(config)# router ospf 10
(config-router)# redistribute bgp tag 355
```

CLI Usage

The OSPF domain can have multiple domain IDs. Configure one as the primary domain ID and the remaining as secondary domain IDs. The primary domain ID is sent to the remote PE with the BGP extended community attributes.

If a secondary domain ID is configured without a primary domain ID, an error occurs, and this message is displayed: “configure primary ID first”.

The domain ID value is NULL in two cases:

- When the domain ID is configured as NULL through the CLI
- When no domain ID is configured, its default value will be NULL

When the domain ID chosen is NULL, OSPF checks the previous configuration of domain ID values in the list. If present, an error occurs, indicating: “non-zero domain-id values exists”. Otherwise, the primary domain ID is set to NULL.

For example: When router PE1 receives a route from remote router PE2, with domain ID, d1, the PE1 router compares this route’s domain ID, d1, with the set of domain IDs present with its OSPF instance. If it matches, it sends the route as the type specified in the Route-Type extended-community. If it does not match, it sends the route as Type-5 LSA.

The `domain-id` CLI should be configured before bringing up an OSPF-VRF session with the CE router. In this way, the OSPF routes received from the CE are stored in NSM, along with domain ID value. In turn, when BGP redistributes these OSPF routes into the VPN cloud, it takes the routes along with the domain ID.

CHAPTER 6 Passive Interface

The passive interface feature simplifies the configuration of distribution routers and allows network managers to obtain routing information from the interfaces in large Internet Service Provider (ISP) and enterprise networks.

Overview

The `passive-interface` command can put all interfaces or a particular interface into passive mode. If no interface name or IP address is used with this command, all interfaces are put in passive mode. If an interface name or IP address is used with this command, a particular interface is put in passive mode. The `no` form of this command removes all interfaces from passive mode if no interface name or IP address is specified, or removes a particular interface from passive mode if an interface name or IP address is specified. For details on this command, the *Open Shortest Path First Command Reference*.

Issue the `passive-interface` command to stop routers from becoming OSPF neighbors on a particular interface. In ISP and large enterprise networks, many of the distribution routers have a large number of interfaces. Configuring passive interface on each of the interfaces can be difficult. The `passive-interface` command (without any interface name) can be used to solve this problem by configuring all the interfaces as passive using a single command.

System Architecture

The following provides a configuration example, then describes configuration without and with the default option for passive interface.

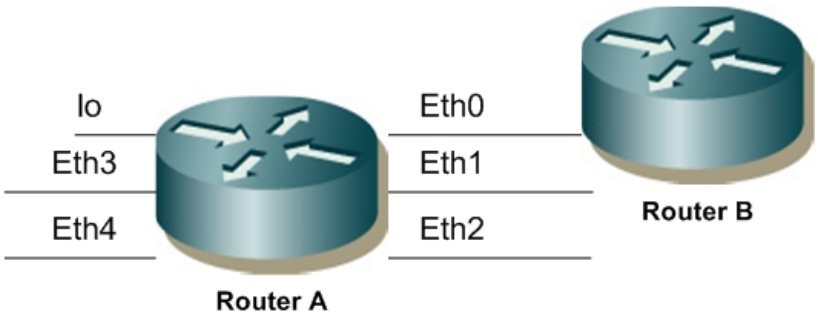


Figure 6-1: Default Passive Interface

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

Without Default Passive Interface

Before the default passive-interface feature, there were two methods to obtain routing information of connected interfaces of Router A:

- Configure OSPF on Router A and redistribute connected interfaces. As a result, a large number of Type-5 LSAs can be flooded over the domain.
- Configure OSPF on all interfaces and manually set most of them as passive. As a result, large Type-1 link-state LSAs might be flooded into the area. The Area Border Router (ABR) creates Type-3 LSAs, one for each Type-1

LSA, and floods them to the backbone. It is possible, however, to have unique summarization at the ABR level, which injects only one summary route into the backbone, thereby reducing processing overhead.

The solution to this problem was to configure OSPF on all interfaces and manually set the `passive-interface` command, with the interface name on the interfaces where adjacency was not desired. In the above illustration, the user had to run the `passive-interface` command with the interface name for all interfaces (Eth1, Eth2, lo, Eth3, Eth4), except Eth0.

With Default Passive Interface

With the default passive-interface feature, this problem is solved by setting all interfaces to passive by default using a single `passive-interface` command, then configuring individual interfaces where adjacencies are desired using the `no passive-interface` command with selected interface names or IP addresses.

Command API

For information about the OSPFv2 API functions that support the passive interface feature, see OSPFv2 Passive-Interface API.

CHAPTER 7 Multi-Area Adjacency

Multi-area adjacency provides support for multiple OSPF areas on a single interface.

Overview

This feature allows the link to be considered an intra-area link in multiple areas and be preferred over other higher-cost intra-area paths. This creates an intra-area path in each of the corresponding areas sharing the same link. For example, an interface can be configured to belong to multiple areas with a high-speed backbone link between two area border routers (ABRs) to allow creation of multi-area adjacencies that belong to different areas.

Features

ZebOS-XP supports the multi-area adjacency features listed below.

- multi-area-adjacency CLI
- ABRs can establish multiple adjacencies belonging to different areas
- Multi-area-adjacency configuration and neighbor discovery as defined in section 2.1 of RFC 5185
- Packet transmission and reception as described in sections 2.2 and 2.3 of RFC 5185
- Configuration error if the link is configured as both virtual link and multi-area-adjacency
- Display of the multi-area-adjacency for multiple areas on the interface

System Overview

OSPF allows a single physical link to be shared by multiple areas to create an intra-area path in each of the corresponding areas sharing the same link.

The following provides a sample configuration and information on multi-area adjacency configuration.

Configuration Example

The following example uses a high-speed link between two ABRs in multiple areas.

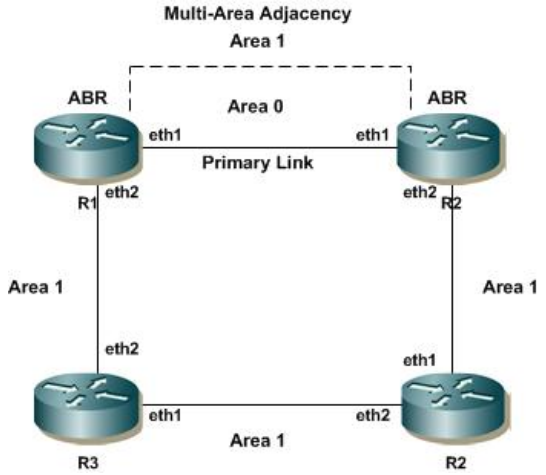


Figure 7-1: Multi-area Adjacency

The backbone-area link between R1 and R2 is a high-speed link. It is desirable to forward Area 1's traffic between R1 and R2 over the high-speed link. Previously, intra-area paths were preferred over inter-area paths. As a result, R1 would always route traffic to R4 through Area 1 over the lower-speed links. R1 would even use the intra-area Area 1 path though R3 to get to Area 1 networks connected to R2.

To reach from R1 to R2 on a direct link, the link is also configured in Area 1. With multi-area adjacency, the existing link is allowed for Area 0 and Area 1. This Area 1 link is configured as multi-area-adjacency and becomes an intra-area path for R1 to directly reach R2. In this way, R1 can directly route the Area 1 traffic from R1 to R2 on the Area 1 link.

Configuration Techniques

Multi-area adjacency establishes adjacency between the ABRs. A specific interface of the area border router (ABR) is associated with multiple areas.

Each multi-area adjacency is announced as a point-to-point link in the configured area. The point-to-point link provides a topological path for that area. Because multi-area adjacency is based on the primary adjacency, the primary adjacency should be up to establish multi-area adjacency.

Multi-area adjacencies are configured between two routers with a common interface. The neighbor address of each multi-area adjacency must be configured.

There is no restriction on multi-area adjacency for the type of areas on which it can be configured. Multi-area adjacency can be configured in backbone and non-backbone areas. However, if a virtual link is configured between two routers through a transit area, a multi-area-adjacency link cannot be configured in a backbone area for a link between these routers in a transit area. Similarly, if a multi-adjacency link is configured between a pair of routers in a backbone area using a primary link in a non-backbone area, a virtual link can not be established between these routers for a transit area the same as the area ID of the non-backbone area of the primary link used for the multi-adjacency link.

System Architecture

OSPF Interface Structure for Multi-Area Adjacency

Multi-area adjacency allows the OSPF interface creation for multiple areas for each interface address. The Interface State Machine (IFSM) is maintained on the respective OSPF interface.

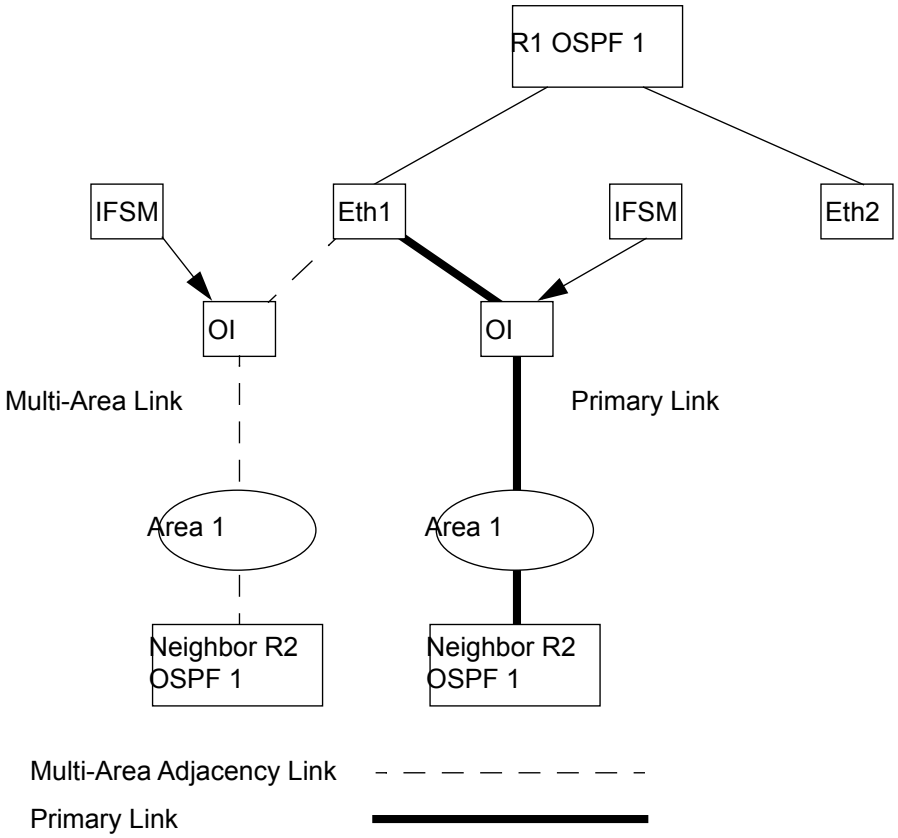


Figure 7-2: Multi-Area Adjacency Interface Structure

In the diagram above, R1 and R2 are ABRs (border of Area 0 and Area 1) running an OSPF instance between them. There is a backbone area high-speed link between R1 and R2. This link is also configured as a multi-area link in Area 1. A separate OSPF interface data structure (OI) is maintained for both the primary and multi-area adjacent links. An IFSM is maintained separately for each OI.

Primary and Multi-Area Adjacent OSPF Interface

Packets associated with multi-area adjacency use the primary link's area as the transit area. Therefore, if the primary interface goes down, the multi-area adjacency corresponding interfaces using the primary interface also go down. Also, if any properties associated with the primary interface change, the same properties are applied to all multi-area adjacency interfaces associated with the primary interface.

If the primary adjacency is disabled (using the `no network` command), the multi-area adjacency is also deleted.

CHAPTER 8 Constrained Shortest Path First

This chapter describes the basic architecture of the Constrained Shortest Path First (CSPF) module. The CSPF module is a library that is linked to either the OSPF-TE module or the IS-IS-TE module.

The CSPF algorithm calculates an optimum explicit route (ER), based on the specified constraints, using the TED (Traffic Engineering Database) and pre-existing Label Switched Paths (LSP). The resulting ER is used by a signaling protocol (RSVP-TE or CR-LDP) to set up LSPs.

To set up traffic-engineered LSPs, ZebOS-XP requires these modules:

- IGP-TE (IGP (interior gateway protocol) routing protocol which supports traffic engineering extension (ZebOS-XP uses OSPF-TE)
- TED (traffic engineering database)
- CSPF

Architecture

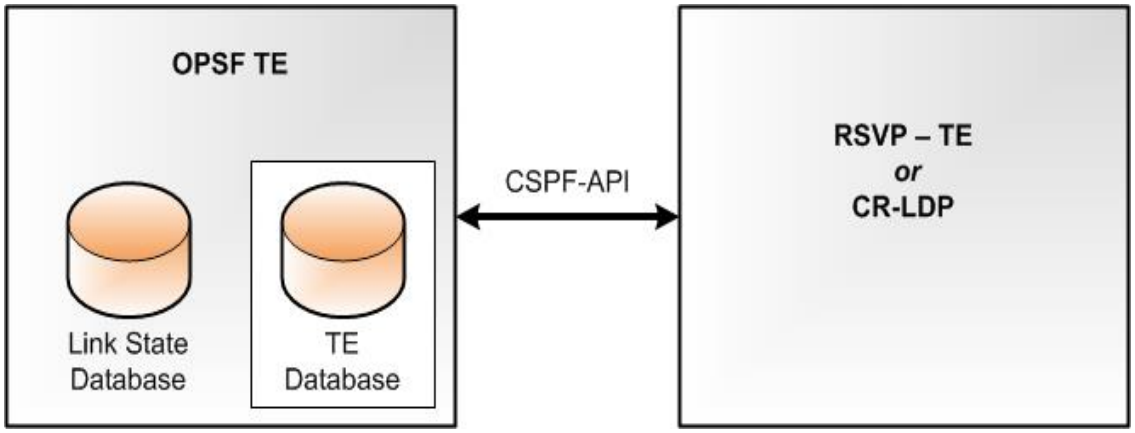


Figure 8-1: CSPF relation to OSPF

IGP-TE

The algorithm to calculate the CSPF requires router data, network data, and traffic engineering data for each link. For OSPF, Type-1 (Router-LSA), Type-2 (Network-LSA) or Type-10 (Opaque-LSA with TE information), fill these roles respectively.

In IGP-TE (OSPF or IS-IS), routers exchange the following data and store it in TED:

- Link type (1 octet)
- Link ID (4 octets)
- Local interface IP address (4 octets)
- Remote interface IP address (4 octets)
- Traffic engineering metric (4 octets)
- Maximum bandwidth (4 octets)

- Maximum reservable bandwidth (4 octets)
- Unreserved bandwidth (32 octets)
- Resource class/color (4 octets)

Traffic Engineering Database

Link attributes are exchanged between neighboring OSPF routers using OSPF-TE with Opaque LSA extensions. To calculate CSPF based LSP, router and network information are needed. In the case of OSPF-TE, Router-LSA and Network-LSA are used.

LSP Attributes

The following attributes comprise an LSP:

Egress address. Specifies the address of the egress endpoint of the LSP tunnel: the interface address or the router id. of the egress router.

Ingress address. Specifies the address of the ingress endpoint of the LSP tunnel. If no address is specified, the router. id of the local router is used. This attribute is optional

Bandwidth. Specifies the bandwidth (in bps) to be reserved for the LSP. .

Hop-Limit. (optional) It specifies the maximum hop count between the ingress and egress routers (including egress). . The value ranges from 1 to 255 (1 meaning the LSP tunnel end-points are directly connected). The default hop limit is 255. This attribute is optional.

Priority. Control LSP setup and pre-emption. IP Infusion Inc. recommends that these two values be the same. .

- **Setup Priority**. Sets the relative importance of LSP setup. This attribute sets the order in which CSPF computes the paths for a given set of LSPs. The valid range of values 0-7 (0 is the highest priority). It also sets whether the setup of a new LSP preempts the setup of an LSP already in the queue; an LSP with a higher setup priority preempts LSPs with lower priorities. The default value of 7 means that this LSP cannot preempt the setup of other LSPs. This attribute is optional.
- **Hold Priority**. Determines the priority of an LSP session for holding resources. Hold priority is useful for deciding whether an existing LSP can be preempted by a new LSP. The value ranges from 0 to 7 where 0 is the highest priority level. The default value 0 means that no LSP can preempt this LSP. This attribute is optional.

Administrative Group. (optional). Specifies resource class filters to enforce admission control policies. Each resource. (link) is assigned one or more class (color) attributes. The class filters can be of two types:

- **Include Filters**. Specify all groups to include while computing CSPF routes. Only links which match the specified color are included in path computation.
- **Exclude Filters**. Specify the groups to exclude from path computation.

  These filters are represented by a 32-bit vector (one each for include and exclude).

Path Attribute. (optional). Specifies the routers included in CSPF route computations. Each router address is . designated as either *strict* or *loose*. A *strict* address in a path signifies (default) that there is no other router between the current router and the specified router. A *loose* address signifies there can be any number of routers between the current router and the specified router.

Tie-Break method. (optional). Tie break method is used to choose a path among equal cost multipaths. There are . three types of tie-break methods:-

- **Least-Fill** A path which is least-filled is chosen.

- **Most-Fill**. A path which is most-filled is chosen.
- **Random**. A path is chosen at random.

Adaptability. This attribute determines whether an LSP is subject to re-optimization when there is a change in network conditions i.e. new resources become available etc. Re-optimization means that the LSP is rerouted through a better path (defined by some established criteria) in case of network changes. This attribute is disabled by default. There is an associated reoptimization timer which determines how often a CSPF computation is undertaken to discover a new route.

LSP Connection Retry Timer. This timer determines how often CSPF makes an attempt to compute a path for a given LSP in case of route computation failures.

LSP Connection Retry Count. This count sets an upper limit to the number of times CSPF does path computation for a given LSP in case of route computation failures.

Shared LSP. This attribute specifies a set of LSPs sharing resources with the LSP for which route computation is desired. This set of LSPs must have the same setup and hold priorities. CSPF takes all such LSPs into consideration while computing a desired route and ensures that on shared links the bandwidth is not counted twice. This helps in conservation of network resources on shared links.

TE Class. This attribute is only available when the Diffserv-TE feature is enabled. It specifies the TE class of the LSP for which a path computation is desired. This feature must be present for successful route computation for DS-TE enabled routers.

CSPF Communications

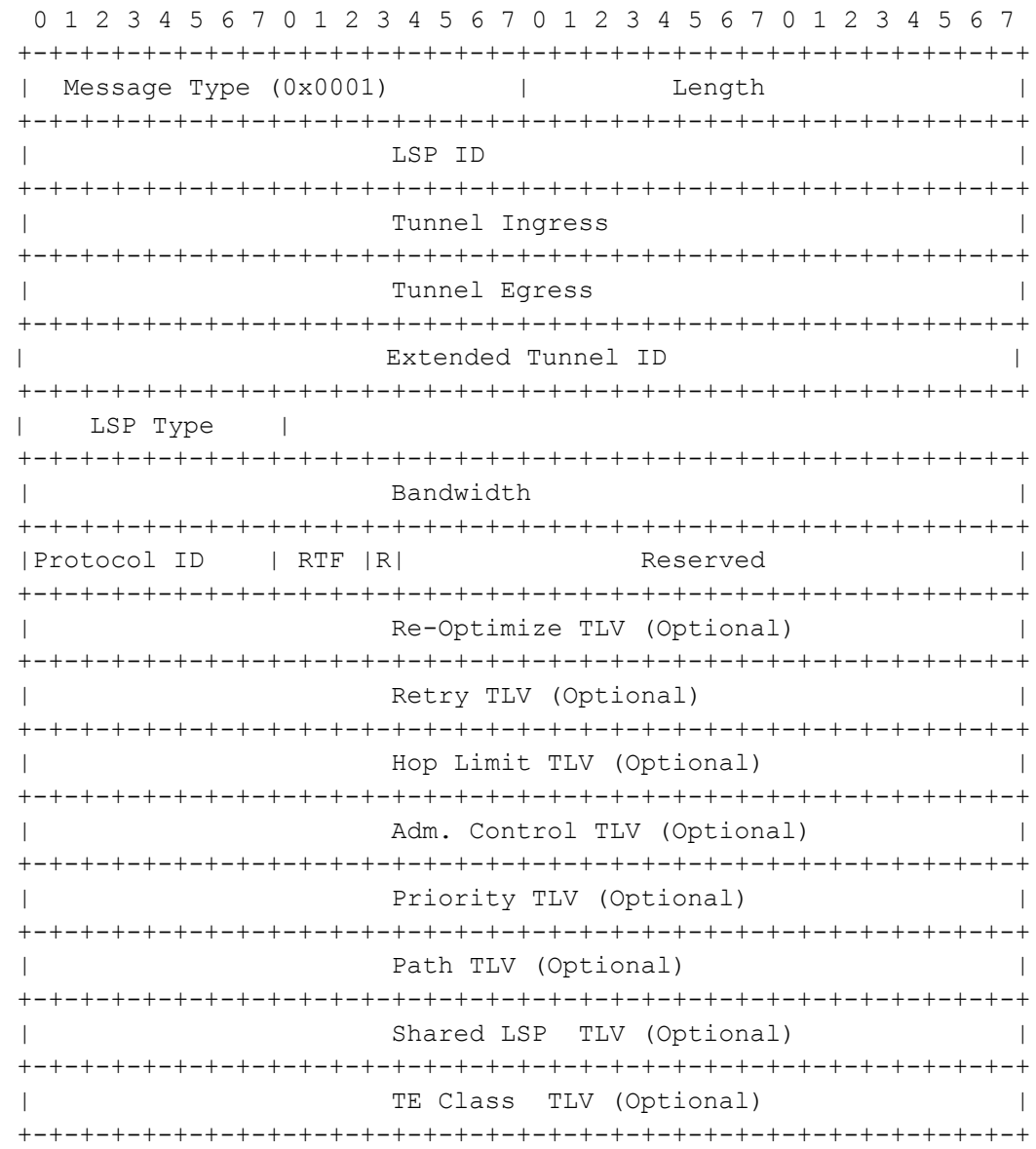
CSPF communicates with a *signaling module* (RSVP-TE/CR-LDP) using the following messages. All the messages are aligned on 32-bit boundaries. The following message structure definitions stipulate how to structure communications with the CSPF module. ZebOS-XP provides no functions for this purpose.

Route Request Message

This message is sent from a signaling module to CSPF to request a route computation based on a set of specified constraints. CSPF returns a Route Message is returned when the calculation can be completed; it returns a Notification message when an error occurs. This message should be sent under the following conditions:

- Compute a new route
- Re-compute an old route (possibly with changed attributes)
- Update route attributes that may lead to route recomputation

Message Structure



Field Descriptions

Message Type. This two-byte, unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types*

Length. This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional (type length value) TLV objects.

LSPID. This four-byte unsigned integer is generated by the signaling module to uniquely identify a route request. This value is used to change attributes, request route re-computations and delete given LSPs. This value is also used by CSPF to respond to route requests and for any route updates for a given LSP.

Note: If some signaling module uses 2 octets to identify LSPs, then it should use implementation specific padding bytes to generate a four octet value.

Ingress. This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel. It defaults to the router ID if no value is specified.

Egress. This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Note: Care should be taken while using interface addresses to ensure that the interface, corresponding to the egress address, belongs in the same area as the ingress.

Extended Tunnel ID. This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

LSP Type. This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

Bandwidth. This four octet unsigned integer specifies the bandwidth (in bytes per second) as a constraint for path computation.

Protocol ID. This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A, *Message and TLV Types* for protocol ID values.

RTF bits. This is a 3-bit flag which specifies the type of route request as discussed previously. The following types are supported.

- **New Route Request**. This specifies a new route computation. The RTF bits should be set to 001.
- **Route Re-computation Request**. This specifies that a re-computation is desired for a pre-existing LSP. It can also be used in conditions where attributes need to be changed for an existing LSP. If no new attribute values are specified, then existing attributes will be used for route computation. To delete an attribute, the corresponding TLV should be specified with the length field set to zero and no TLV data. The RTF bits should be set to 010.
- **Update priority** –

Note: Flag bits should be set appropriately for each condition. A signaling module expects either a route message or a notification message (in case of computation failure or other exceptional conditions).

R bit. This bit specifies the resilience attribute of the LSP. It should be set to 1 if path resilience is required.

Reserved.This 20-bit field should be set to zero.

Note: The remaining fields are all optional; they may be included in the message in any order.

Re-Optimize TLV. This is an optional TLV which indicates that route re-optimization feature is required for the given LSP. A re-optimization timer value can be specified for CSPF path recomputation; otherwise a default value is used.

Retry TLV. This is an optional TLV which specifies the number of times CSPF should try to compute a route for the given LSP in case of route failure. It also specifies the interval between successive attempts to compute routes. The TLV length should be set to zero and no TLV data should be set if default values for retry interval and retry limit are desired. See section TLVs for encoding.

Hop Limit TLV. This optional TLV specifies the number of maximum hops that the LSP can traverse. This hop limit does not include ingress router. See section TLVs for encoding.

Adm. Control TLV. This optional TLV specifies the set of resource classes to be included and excluded for path computation. See section TLVs for encoding.

Priority TLV. This optional TLV specifies the setup and holding priority of the given LSP. See section TLVs for encoding.

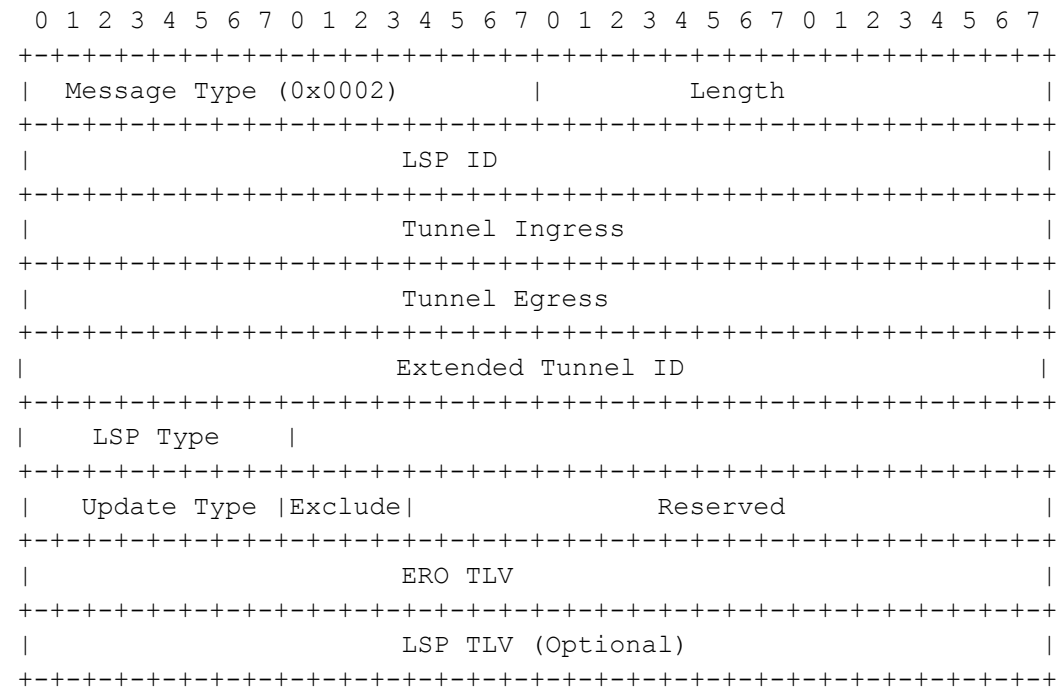
Path TLV. This optional TLV specifies the routers that must be included in the CSPF path computation. See section TLVs for encoding.

Shared LSP TLV. This optional TLV specifies a list of LSPs sharing network resources with the LSP for which a route computation is desired. Refer to the CSPF Communications section for encoding.

TE Class TLV. This optional TLV is only available when Diffserv-TE feature is enabled. It specifies the TE class of the given LSP. See the CSPF Communications section for encoding.

Route Message

This message is generated by CSPF module in response to a route request message, if a route is successfully computed. It contains the explicit path to be used for LSP setup.



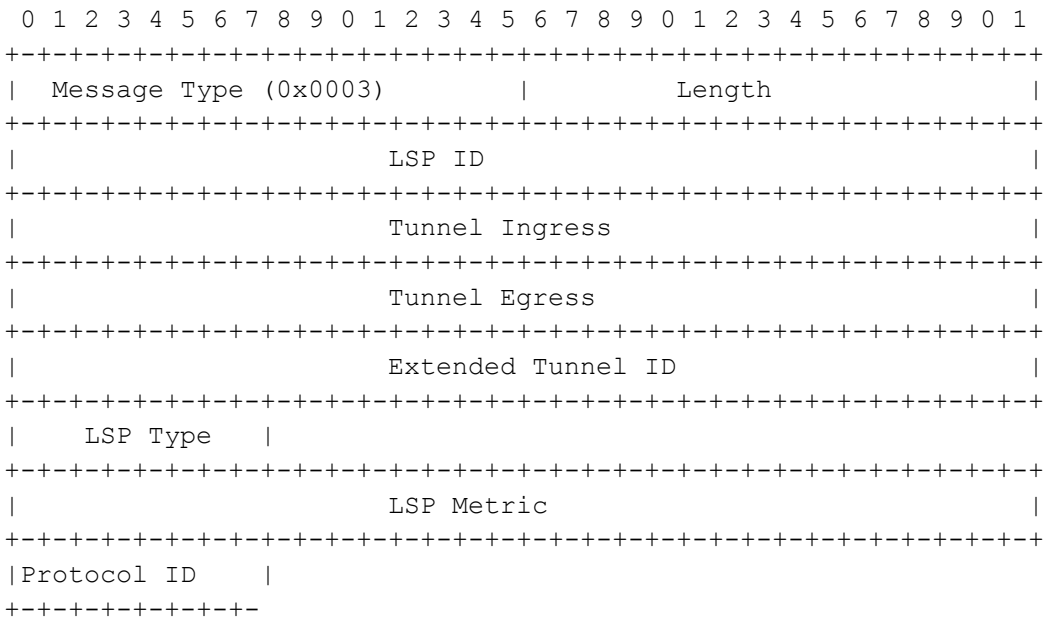
- **Message Type** – This two-byte unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types* for message types.
- **Length** – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.
- **LSPID** – This four-byte unsigned integer identifies the route request message corresponding to this route message.
- **Ingress** – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.
- **Egress** – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.
- **Extended Tunnel ID** - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.
- **LSP Type** - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.
- **Update Type** – This is a 1 byte field which can take the following values:-
  - **Route Response** – 0x00 This indicates that the route message is in response to a route request message.
  - **Route Optimization** – 0x01 This indicates that the route message is due to automatic route optimization performed by CSPF.
  - **Reroute on Failure** – 0x00 This indicates the existing path for the LSP is not feasible anymore. CSPF checks the feasibility of a given LSP route if the resilience feature is enabled for the same route.

- **Exclude Type** - This nibble specifies whether requested node/link protection is available for the given LSP path computation request.
- **Reserved** – This three-byte field should be set to zero.
- **ERO TLV –** This TLV specifies the explicit path for LSP setup. This path has been calculated based on the specified constraints. See section “TLVs” for encoding.
- **LSP TLV** – This is an optional TLV which specifies one or more LSPs which need to be pre-empted for the current LSP setup to be completed. See section “TLVs” for encoding.

  Note: The signaling module may use protocol specific methods to preempt LSPs and create the new LSP or may employ other methods for setting up a new LSP.

LSP Established Message

This message is sent to CSPF to confirm the setup of a signaled LSP. This message should be sent for LSPs which are setup using CSPF (for computation) as well as LSPs which are manually setup (without using CSPF).



Message Type – This two octet unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types* for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer is generated by the signaling module. This should be the same as the LSPID used in corresponding route request message (for CSPF based LSP setup) or it should be a new unique 32-bit value generated by the SM.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

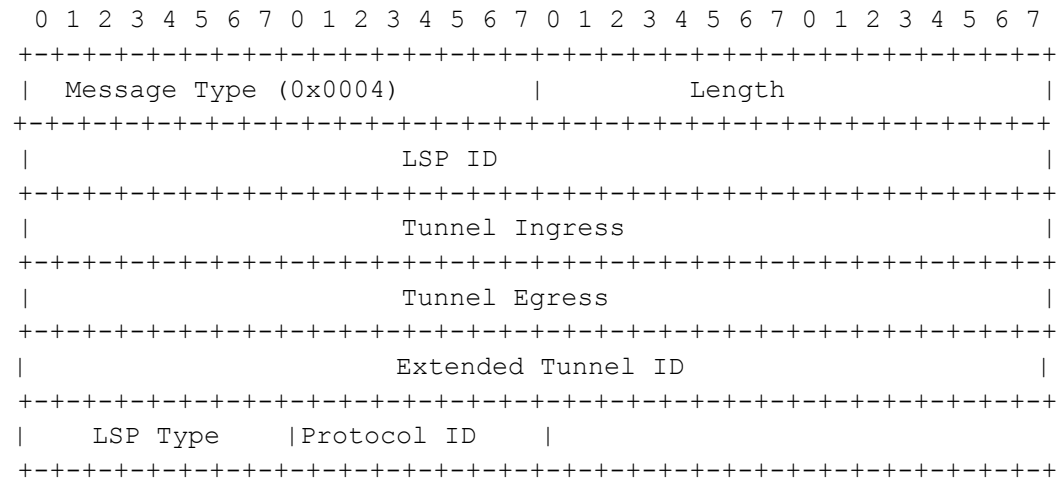
LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

LSP Metric – This four-byte unsigned integer is used to specify a TE metric for this LSP. This metric may be different from OSPF cost metric.

Protocol ID – This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A, *Message and TLV Types* for protocol ID values.

LSP Delete Message

This message is sent to CSPF to delete a given LSP (identified by LSP ID).



Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types* for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer identifies the LSP to be deleted.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

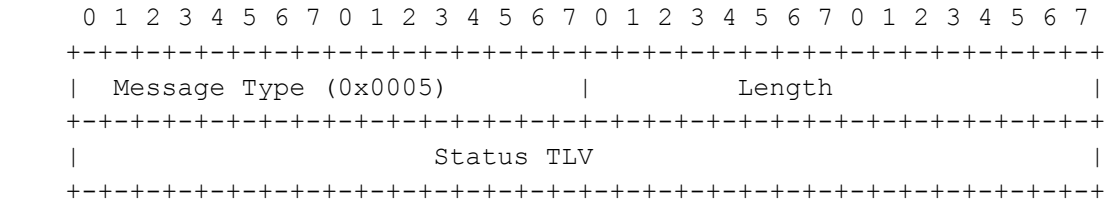
LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

Protocol ID – This identifies the signaling protocol (RSVP-TE or CR-LDP). Refer to Appendix A, *Message and TLV Types* for protocol ID values.

Reserved – This three-byte field should be set to zero.

Notification Messages

These messages are exchanged between CSPF and signaling module to indicate error or other exceptional conditions. The general format for a notification message is given below.



Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types* for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

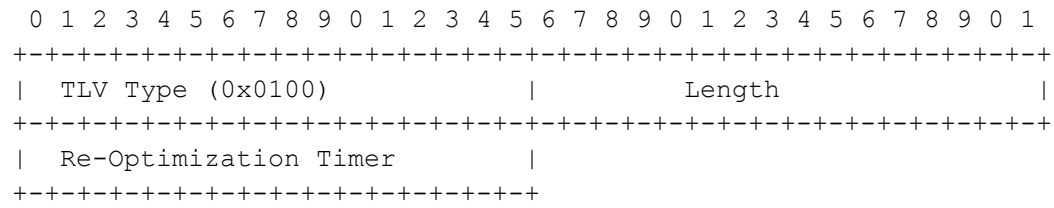
Status TLV – This TLV describes the type of notification and associated data. See section “TLVs” for encoding.

TLVs

The following TLVs can be used in the messages exchanged between CSPF and signaling modules.

Re-Optimize TLV

This TLV specifies whether re-optimization feature is enabled for a given LSP.



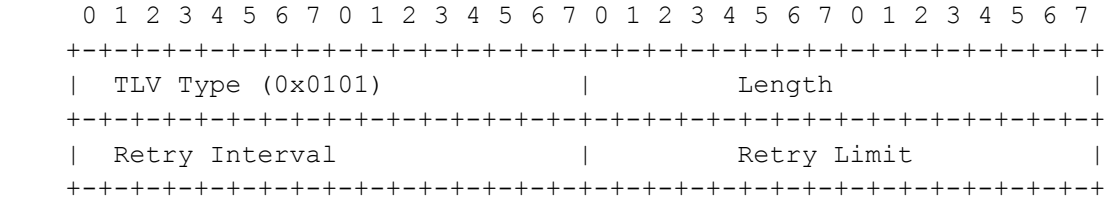
TLV Type - This two octet unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

Length - This two octet unsigned integer specifies the length of data excluding the TLV type and length field.

Re-optimization Timer - The two octet unsigned integer specifies the time interval (in seconds) between successive CSPF route computation for LSP route optimization.

Retry TLV

This TLV specifies whether CSPF should perform route computation again in case of computation failures and associated retry parameters detailed below.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

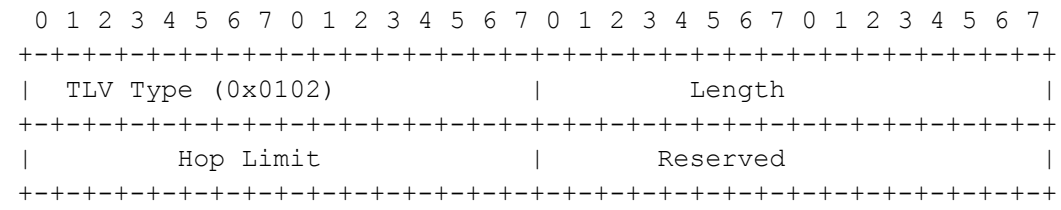
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Retry Interval – This two-byte unsigned integer specifies the time interval (in seconds) between successive route computations in case of computation failure.

Retry Limit - This two-byte unsigned integer specifies the number of times route computations should be done in case of computation failure.

Hop Limit TLV

This TLV specifies the constraint on the number of maximum hops that a computed route can have. This hop limit excludes the ingress router.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

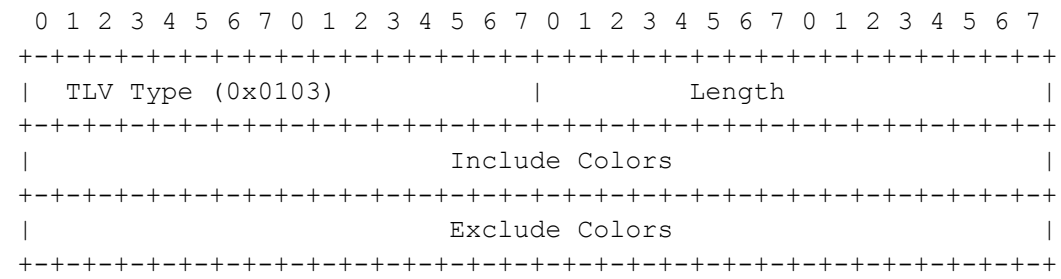
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Hop Limit – This two-byte unsigned integer specifies the constraint on the number of maximum hops that a computed route can have. This hop limit excludes the ingress router.

Reserved – This two-byte field should be set to zero.

Admission Control TLV

This TLV includes admission control information which specifies the set of link colors which need to be included and excluded from route computation.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

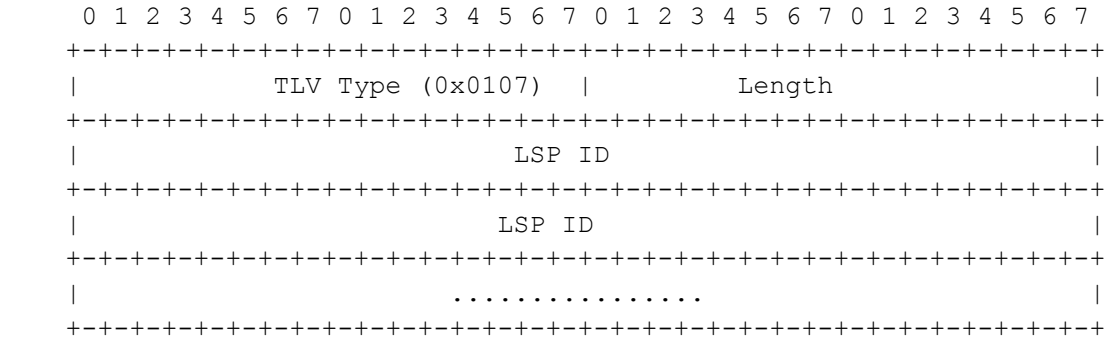
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Include Colors – This two-byte unsigned integer specifies the class (color) of resources that must be included in CSPF route computation. Each color is represented by a bit position in the 32-bit mask.

Exclude Colors – This two-byte unsigned integer specifies the class (color) of resources which must be excluded in CSPF route computation. Each color is represented by a bit position in the 32-bit mask.

LSP TLV

This TLV contains list of LSPs. It can be used to convey a list of LSPs which need to be pre-empted.



`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

`LSPID` – LSPID of an LSP

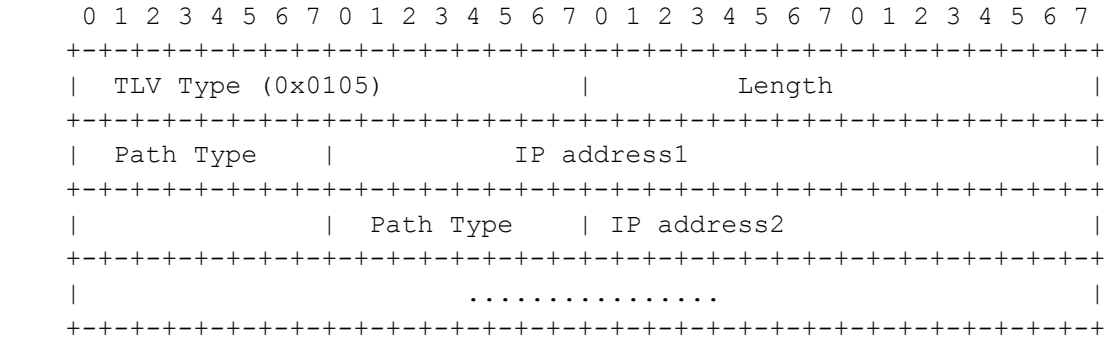
The remaining fields are variable in length.

Shared LSP TLV

This TLV specifies the set of LSPs sharing network resources with the LSP for which a route computation is desired. Refer to the section on LSP TLV for packet details.

Path TLV

This TLV specifies a list of IP addresses which should be included in the route computation in the specified order. Each address can be specified as strict or loose.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

Length – This two-byte unsigned integer specifies the number of IP addresses in the following data. The actual size of TLV data can be calculated by the formula

size = length * 5

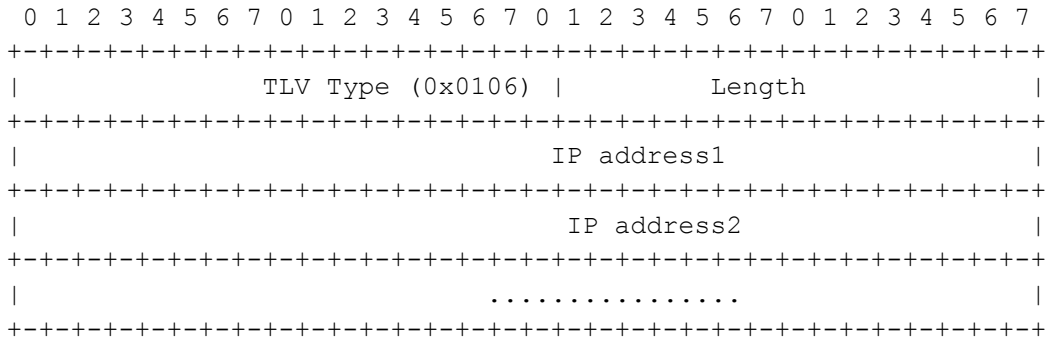
Path Type – Strict or loose

IP address – IP address of node to be considered.

The remaining fields are variable in length.

ERO TLV

This TLV contains a list of IP addresses which form an explicit route from ingress to egress (including the egress router IP address).



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

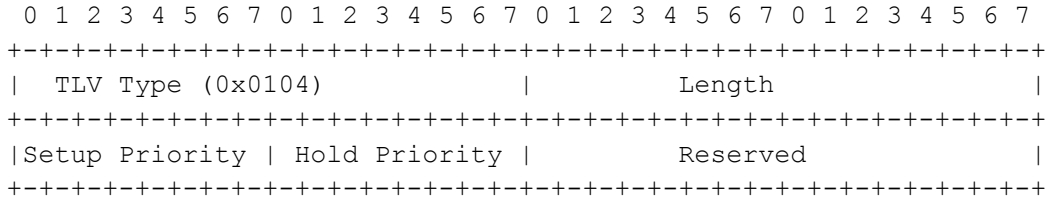
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

IP address – IP address of next router.

The remaining fields are variable in length.

Priority TLV

This TLV specifies the setup and hold priorities of the LSP. Each priority can take a value from 0 to 7.



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

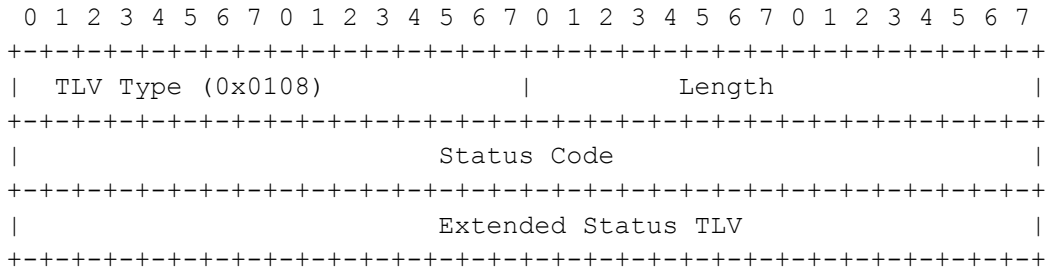
Length – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

Setup Priority – Setup priority of LSP.

Hold Priority – Hold Priority of LSP.

Status TLV

This TLV carries status information for error or exceptional conditions.



`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

`Status Code` – This four-byte unsigned integer specifies the status code for this TLV. The following status codes are defined:-

- SHUTDOWN - This is used to signal to the receiver that the sender is terminating the tcp link.
- ROUTE NOT FOUND - This is used by CSPF to tell the signaling module that the CSPF route computation failed for a given LSP.
- ROUTE FAILURE - This is used by CSPF to notify the signaling module about failure of a route for a given LSP.
- PACKET DECODE FAILURE - This can be sent by either signaling module or CSPF module on receipt of an invalid CSPF message (through a route request etc.).
- REQUEST MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing route request message.
- DELETE MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing LSP delete message.
- ESTABLISHED MSG PROCESSING FAILURE - This is sent by CSPF to signaling module if there is an error encountered in processing LSP established message.

Shared LSP TLV

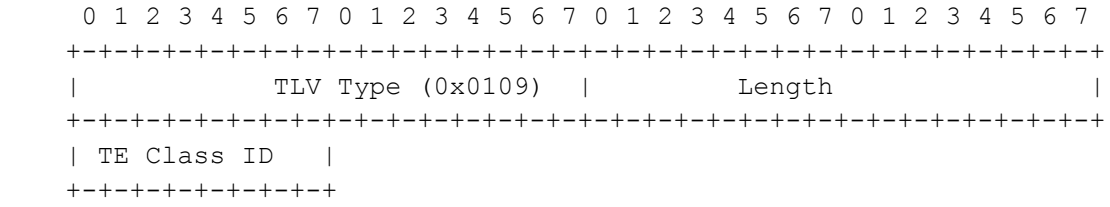
This TLV specifies the set of LSPs sharing network resources with the LSP for which a route computation is desired.

Please refer to the section on LSP TLV for packet details.

TE Class TLV

This TLV is available only if DS-TE feature is enabled.

It specifies the TE class identifier of a given LSP.



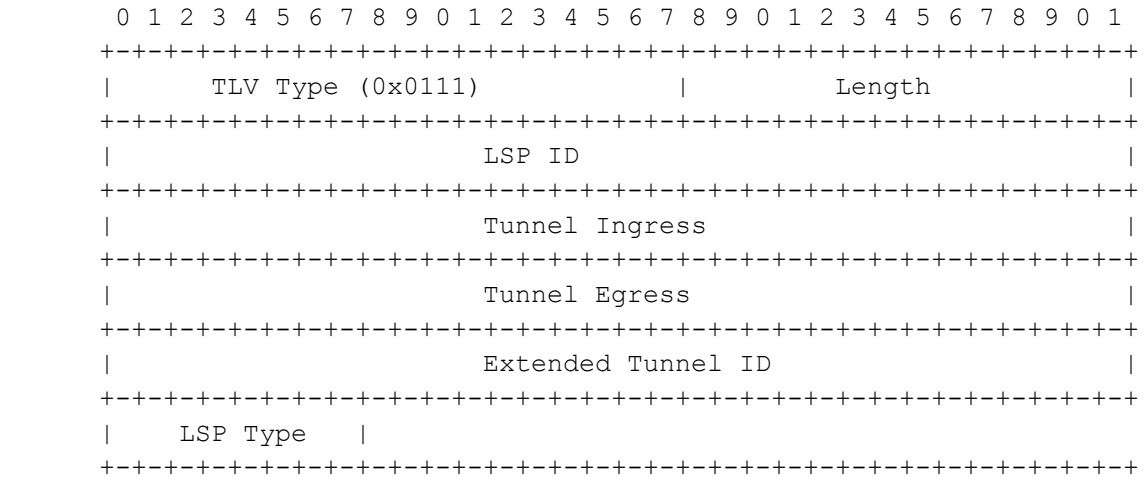
`TLV Type` – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

`Length` – This two-byte unsigned integer specifies the length of data excluding the TLV type and length field.

`TE Class ID` – TE class identifier for an LSP.

Extended Status TLV

This TLV carries additional status information for error or exceptional conditions. This is used to augment the information carried by the Status TLV.



Message Type – This two-byte unsigned integer identifies the type of message. Refer to Appendix A, *Message and TLV Types* for message types.

Length – This two-byte unsigned integer specifies the length of data excluding the message type and length field. It takes into account all the required fields as well as the optional TLV objects.

LSPID – This four-byte unsigned integer identifies the LSP.

Ingress – This four octet unsigned integer identifies the address of the local router that is used to create an LSP tunnel.

Egress – This four octet unsigned integer specifies the address of the tunnel endpoint. It could be set to the router ID of the egress router or one of the interface addresses.

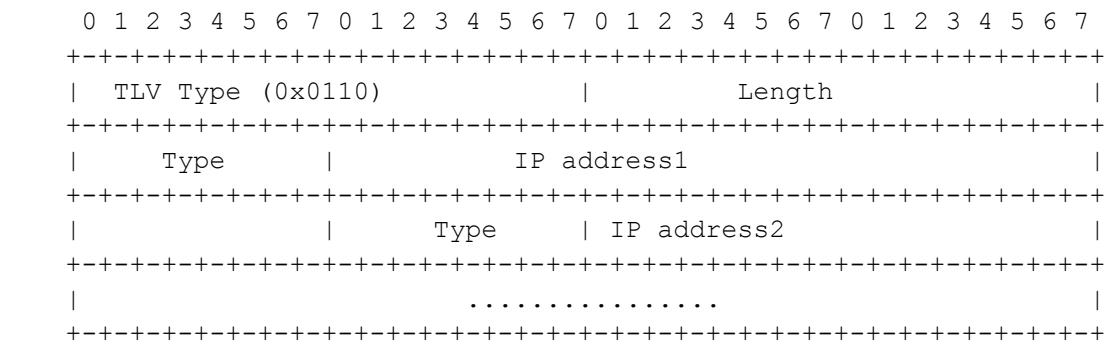
Extended Tunnel ID - This four octet unsigned integer specifies the extended tunnel identifier used by signalling protocols.

LSP Type - This octet specifies a primary or a backup LSP. The value 0 signifies primary LSP and the value 1 signifies backup LSP.

CSPF Modifications for RSVP-TE Fast Reroute

For the RSVP-TE Fast Reroute feature, CSPF has been extended to support another path constraint, the exclude route. A new TLV (Exclude Path) has been added to route request message. This TLV contains a set of IPV4 or IPv6 addresses which should be avoided by CSPF computation engine while computing a constrained route. Each address also contains a flag specifying whether a link on a node or the node itself is to be excluded.

IPv4 Exclude Path TLV



TLV Type – This two-byte unsigned integer identifies the type of TLV. Refer to Appendix A, *Message and TLV Types* for TLV types.

Length – This two-byte unsigned integer specifies the number of IP addresses in the following data. The actual size of TLV data can be calculated by the formula

size = length * 5

Path Type – Exclude Route Link and Exclude Route Node

- Exclude Route Link - The presence of this path type indicates that the link identified by the specified IP address should be avoided.
- Exclude Route Node - The presence of this path type indicates that the node, to which the specified IP address belongs, should be avoided by the route computation engine. This the default value.

IP address – IP address of node to be considered.

The remaining fields are variable in length.

OSPFv3 CSPF

ZebOS-XP supports traffic engineering extensions to OSPF Version 3 (OSPFv3), and TLVs and sub TLVs for Intra-Area-TE Link State Advertisement (LSA) used in constrained-based SPF calculation.

TLVs and sub TLVs extend TE capabilities to IPv6 networks. The functions defined in RFC 4203 and RFC 3270 are extended to OSPFv3 by using the TLVs and sub-TLVs described in the IETF draft, draft-ietf-ospf-ospfv3-traffic-09.txt.

Intra-Area TE LSA

The Intra-Area-TE LSA payload consists of many TLVs and sub TLVs that can be configured to propagate resource availability information in link-state routing updates. This information is used to perform a constraint-based SPF (CSPF) calculation.

To accommodate OSPFv3, a link-state (LS) type is defined for the Intra-Area TE LSA, with function code 10. The LSA payload consists of one or more nested TLV triplets. There are two applicable top-level TLVs:

- 2 - Link TLV
- 3 - Router IPv6 Address TLV

The Router IPv6 Address TLV advertises a reachable IPv6 address. This stable IPv6 address is always reachable if there is connectivity to the OSPFv3 router.

The Link TLV describes a single link and consists of a set of sub TLVs. All of the sub TLVs in RFC 3630, other than the Link ID sub TLV, are applicable to OSPFv3. The Link ID sub TLV is not used in OSPFv3, due to the protocol differences between OSPFv2 and OSPFv3. Thus, ZebOS-XP uses three sub TLVs for the Link TLV:

- 18 - Neighbor ID (8 octets)
- 19 - Local Interface IPv6 Address (16N octets, where N is the number of IPv6 addresses)
- 20 - Remote Interface IPv6 Address (16N octets, where N is the number of IPv6 addresses)

Differentiated Services

The sub TLVs defined in RFC 4203 and RFC 3270 extend to OSPFv3. The following sub TLVs are added to the Link TLV:

Sub TLV Type	Length	Name
11	8	Link Local/Remote Identifiers
14	4	Link Protection Type
15	Variable	Interface Switching Capability Descriptor
16	Variable	Shared Risk Link Group

Traffic Engineering Database

The Traffic Engineering Database (TED) stores the traffic engineering and resource availability information. The algorithm to calculate the CSPF requires router, network, and traffic engineering data for each link. For OSPFv3, Type-1 (Router-LSA), Type-2 (Network-LSA) or Type-10 (Inter-Area TE LSA), fill these roles, respectively.

MPLS LSP Destination Route Deletion

RSVP-TE uses the CSPF module to compute the Explicit Route Object (ERO) for its LSPs. CSPF uses the TE Link State Database (LSDB) populated by the OSPF module in calculating a constraint-based path from the ingress to the egress.

The MPLS LSP Destination Route Deletion feature defines a mechanism for the CSPF server to detect the loss of egress reachability and asynchronously communicates this to the RSVP-TE module. This eventually speeds up the LSP tear-down operation, instead of requiring long intervals for refresh timer expiration.

This section describes the procedures involved in detecting route deletions and the notification process between CSPF and RSVP-TE. It also describes the changes in the code in the OSPF-TE, CSPF, and RSVP-TE modules to accommodate this feature.

Design Overview

The OSPF TE database is used to detect LSP path deletion in two scenarios:

- Handling the CSPF established message from the RSVP-TE to CSPF server
- Handling TE link TLV deletion (triggered by OSPF updates) from the TE LSDB

The above scenarios are explained in detail in the subsections which follow.

The CSPF server stores an address to LSP mapping information in its main structure (`CSPF`). Upon TE LSA deletion, OSPF-TE intimates this information to the CSPF server.

The CSPF server notifies the CSPF client, and subsequently the RSVP-TE module, about route changes in the LSP path.

Multiple LSPs may be dependent on the TE link to be deleted. In this case, a single notification is generated with its status TLV containing all the encoded LSP keys. The CSPF_CODE_ROUTE_FAIL status code is used for this notification.

RSVP-TE handles this notification message similarly to the handling of Route Delete messages received from NSM in non-CSPF based LSPs cases. It tears down the LSP by propagating Path Tear messages along the LSP path and sends an LSP Delete message back to the CSPF to update its LSP Table.

LSP Established Messages from RSVP-TE

RSVP-TE sends a Route Request message to the CSPF server to compute an LSP path. CSPF replies to this with a Route message containing the constraint-based hop-by-hop path. The RSVP-TE ingress propagates the PATH message towards the egress, which returns its confirmation with a RESV message. In turn, the LSP is established, and RSVP-TE communicates this establishment to CSPF via an LSP ESTABLISHED message. CSPF receives this message and updates its LSP status.

CSPF creates or updates a binding data structure containing a reference count (`refCount`) that indicates the number of dependent LSPs on this address.

A Patricia tree is keyed on the set of all IP addresses that appear in the LSP path for each LSP. Each node in the tree corresponds to a local address in the `cspf_nexthop_data` structure. The local address is the key because the corresponding local address is always impacted and gets deleted upon link failure or routing changes across the LSP path. Each node stores the `refCount` element that indicates the number of LSPs dependent on the address.

This binding is created only when a link is used for the first time by an LSP. When subsequent LSP(s) use the same link, a search is made for the presence of a node with this address in the Patricia tree. If present, the reference count is incremented.

The following illustration depicts the steps for handling the receipt of an LSP-established message by the CSPF server.

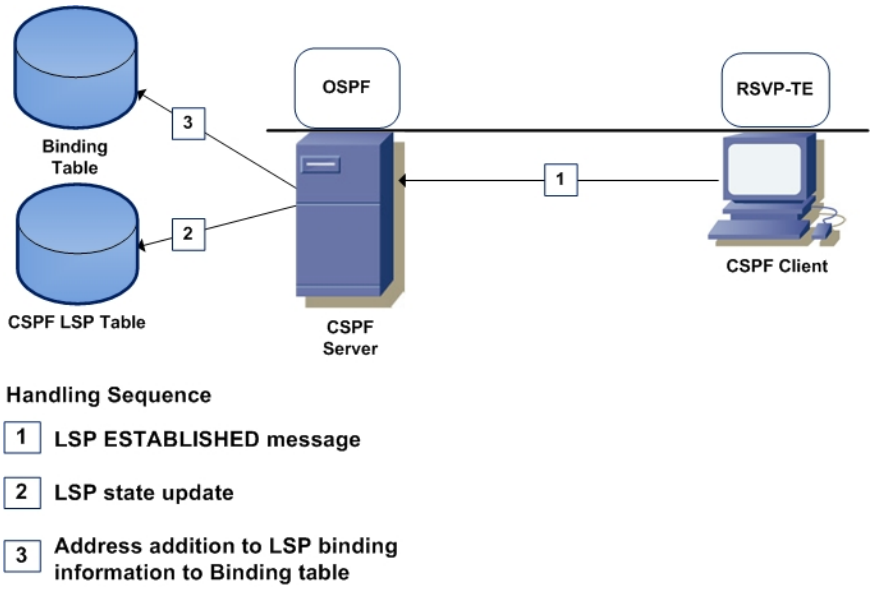


Figure 8-2: Establish LSP with CSPF

Step 3 demonstrates a significant performance optimization. Without this step, it would be required to compare each prefix in the ERO of each LSP in the LSP table against the local interface in the TE Link TLV being deleted. This would be unnecessary overhead, because not every Link TLV deletion may impact the set of established LSPs in CSPF at any point of time.

TE Link TLV Deletion

TE-Link TLV deletion occurs in response to routing changes detected by OSPF. This deletion may be a consequence of a corresponding interface going down or removal of a network from the OSPF domain. It may be attributed to deletion of the entire TE LSA or only the Link TLV in that LSA.

Receipt of TE Link TLV Deletion is detected by the OSPF module section handling TE operation. The OSPF TE module sends the local address in the link TLV to the CSPF server, before deleting it from the TE LSDB. CSPF then checks for the presence of this information and ascertains whether the deletion would impact active LSP(s): It does this by searching its Binding table, based on the address sent and verifies if a matching entry exists. The reference count is extracted from this node and stored along with the key.

The CSPF server scans each node in the LSP table, starting from the top, inspecting LSP(s) in each node for any matching local address in their route list. On each successful match, the reference count is decremented by 1, and the `cspf_lsp_key` of the matching LSP is added to a list. This operation continues until the reference count becomes 0.

When the reference count becomes 0, all of the affected LSPs, whose LSP keys are available as a list, are identified. CSPF sends out a notification message with the status code set to CSPF_CODE_ROUTE_FAIL, and the status TLV containing the LSP keys encoded as per the list. This indicates that one of the hops along the path is now unreachable, and as a result, signals an asynchronous LSP failure for each of those LSPs to RSVP-TE.

RSVP-TE handles this notification type by performing operations to tear down this session, including propagation of the PATH Tear message towards the egress. This propagation can be as far as the last reachable node in the outcome of the routing change that occurred.

The LSP tear-down mechanism includes dispatch of the LSP Delete message back to the CSPF module to update its local LSP table. The following illustration depicts the sequence of events in handling TE LSDB based route deletions.

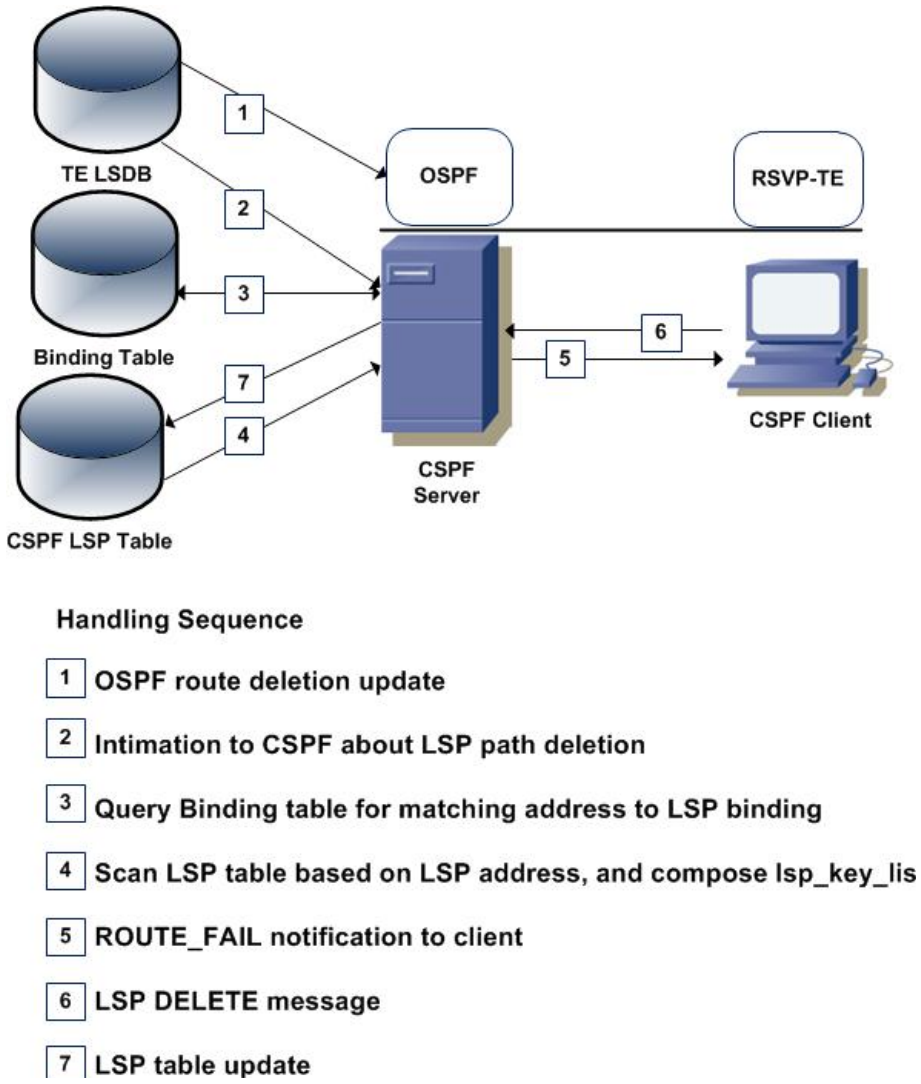


Figure 8-3: Route Delete Processing

Steps 2 through 5 correspond to the MPLS LSP Destination Route Deletion feature.

CSPF Computation Algorithm

The CSPF route computation algorithm is a modified form of SPF algorithm. The steps involved in the computation are the following:

1. Verify that the egress address does not belong to ingress node and ingress address is a valid address on the ingress node.
2. Verify that egress and ingress belong to the same area.
3. Verify that the addresses specified in the path constraint belong to the same area as ingress and are reachable.
4. Verify that the hop limit constraint is not smaller than the number of addresses specified in the path constraint.
5. Perform SPF computation taking into account all the links which satisfy all the specified constraints.

6. If the computation fails, then recompute by adding back the bandwidth associated with LSPs with lower hold priority.
7. A tie between two equal cost paths is resolved in the following manner:
    1. Choose the path with lower hop count
    2. If hop count is same, then choose the path with according to the tie-break method specified. In case of "most fill", a path with lower minimum available bandwidth ratio is chosen. In case of "least fill", a path with higher minimum available bandwidth ratio is selected.
    3. If there is still a tie, then choose a path at random.

        Note: Available Bandwidth ratio = Available bandwidth/Max. Reservable bandwidth.

        Minimum Available Bandwidth Ratio = Least available bandwidth ratio of all the links forming the path.
8. In the case of path constraints, the computation is done in stages. In each stage, the start and end nodes for that stage are chosen from among the ingress, egress and path addresses. For example, in the first stage, ingress is chosen as the start node and the first path address is chosen as the end node. In the second stage, the first path address is chosen as the start node and next path address is chosen as the end node. This continues until the actual egress is chosen as the end node or the intermediate computation fails.

CHAPTER 9 Loop-Free Alternate Fast Reroute

This chapter describes the OSPF Loop-Free Alternate Fast Reroute (LFA-FRR) feature as specified by RFC 5286, hereafter referred to as *fast rerouting*.

Overview

Fast rerouting reduces the failure reaction time when a primary next hop fails. Fast rerouting lets you configure a per-prefix loop-free alternate path that redirects traffic to a next hop other than the primary neighbor. The forwarding decision is made and service is restored without other routers' knowledge of the failure.

Normally in an interior gateway protocol (IGP) such as OSPF, a change in network topology upon a failure triggers a network-wide convergence. For example, when a local link fails, a router:

1. Signals the event to its neighbors via OSPF link-state-advertisements
2. Re-computes new primary next-hops for all affected prefixes
3. Installs those new primary next-hops in the Forwarding Information Base (FIB)

Until the new primary next-hops are installed, traffic directed towards the affected prefixes is discarded. This convergence process can take hundreds of milliseconds on a given router.

With fast rerouting, a router can correct a path failure before the OSPF convergence process completes. Fast rerouting can reduce the amount of time needed to reroute traffic to less than 50 milliseconds, minimizing the packet loss after a failure.

With an alternate path, the packets rejoin the original route downstream from the failure in two different ways:

- If the packets rejoin the original route at the remote node of the protected interface, the alternate path provides *link protection*. Use link protection when you assume that only a single link might become unavailable but that the neighboring router on the primary path would still be available through another interface.
- If the packets rejoin further downstream than the remote node, then the alternate path provides *node protection*. Node protection establishes an alternate path through a different router altogether. Use node protection when you assume that access to a router is lost when a link is no longer available.

In Figure 9-1, router A is using its primary path to reach the destination prefix via Router B, but has also calculated a loop-free alternate path via Router C which it has pre-installed into its FIB.

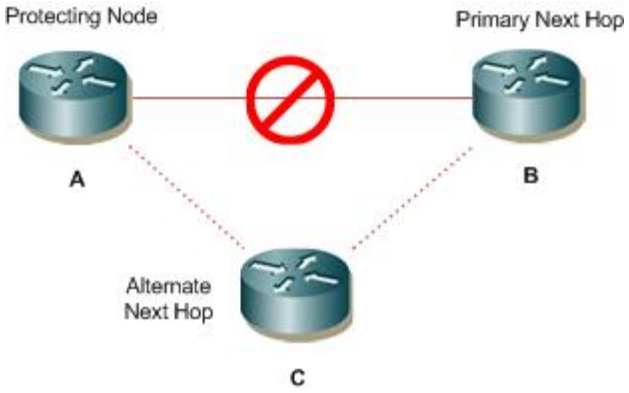


Figure 9-1: Link protection

There is now a link failure between routers A and B. As soon as router A detects the failure, router A begins using its loop-free alternate path via router C to quickly restore traffic flow. This is an example of link protection, where the link between A and B is protected against failure.

Figure 9-1 shows an example of node protection that protects against router B failing.

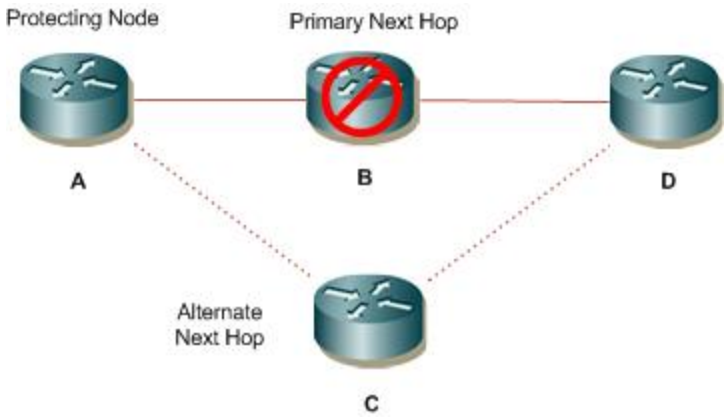


Figure 9-2: Node protection

Types of Repair Paths

When a primary path fails, many paths are possible repair candidates. The fast rerouting feature groups paths into types. The default selection policy prioritizes the types in the following order:

1. primary-path
2. interface-disjoint
3. node-protecting
4. broadcast-interface-disjoint

primary-path

This type of path comes from the Equal-Cost Multipath Path (ECMP) set. An ECMP found during the primary shortest path first (SPF) repair might not be desirable in networks where traffic exceeds the capacity of any single link.

interface-disjoint

This type of path is a point-to-point interface with no alternate next hop for rerouting if the primary gateway fails, thus protecting the interface.

node-protecting

This type of path bypasses the `primary-path` gateway router which might not protect the router that is the next hop in the primary path.

broadcast-interface-disjoint

This type of path is not connected to a broadcast network. Repair paths protect links when a repair path and a protected primary path use *different* next-hop interfaces. However, on broadcast interfaces, if the repair path is computed via the same interface as the primary path, but their next-hop gateways are different, the router is protected but the link might not be.

Configuring Fast Rerouting for OSPFv2

You enable the fast reroute feature for OSPFv2 by calling the `fast-reroute keep-all-paths` command.

Use the `fast-reroute tie-break` command to change the repair path priorities. You can also call a function to reset the repair path priorities to their default values.

You permit an interface to be used as a repair path or prohibit an interface from being used as a repair path with the `ip ospf fast-reroute per-prefix candidate disable` command.

These commands display fast-reroute routes:

- `show ip route fast-reroute`
- `show ip ospf route fast-reroute`

Configuring Fast Rerouting for OSPFv3

You enable the fast reroute feature for OSPFv3 by calling the `fast-reroute keep-all-paths` command.

Use the `fast-reroute tie-break` command to change the repair path priorities. You can also call a function to reset the repair path priorities to their default values.

You permit an interface to be used as a repair path or prohibit an interface from being used as a repair path with the `ipv6 ospf fast-reroute per-prefix candidate disable` command.

These commands display fast-reroute routes:

- `show ipv6 route fast-reroute`
- `show ipv6 ospf route fast-reroute`

CHAPTER 10 On Demand Circuit

This chapter describes the On Demand Circuit feature as specified by RFC 1793 for OSPFv2 and OSPFv3.

Overview

OSPF sends hellos every 10 seconds and link-state advertisements (LSAs) are refreshed every 30 minutes. These functions maintain neighbor relationships and ensure that the link-state databases are up-to-date. However, even this amount of traffic is undesirable on demand circuits.OSPF demand circuit option suppresses hello and LSA refresh functions.There is no periodic exchange of hello or LS updates till there is a change in topology.

Flooding reduction feature for OSPF minimizes traffic created by a periodic refresh of LSAs in OSPF domains with large number of LSAs. Unlike the OSPF demand circuit feature, flooding reduction is usually configured on leased lines. Flooding reduction uses same technique as demand circuits to suppress the periodic LSA refresh.

OSPF sends Hello packets periodically out of all router interfaces. But sending of Hello's even after neighbor has reached Full state is undesirable for demand circuits. These routers set DC-bit in Hello and Database Description packets sent out of the demand interface. Receiving Hello or Database description packets with DC-bit set indicates agreement that no exchange of Hello and Database Description. Receiving a Hello with DC-bit clear or Database description packet with DC-bit clear indicates that the other refuses to suppress Hellos.

Flooding over demand circuits (point-to-point or otherwise) is modified if and only if all routers have indicated that they can process LSAs having DoNotAge set.

Hello suppression is applicable to only point-to-point and point-to-multipoint network types whereas LSA suppression is applicable to all network types.

Configuring Demand Circuit

When ospf demand-circuit is enabled on an interface (point-to-point or point-to-multipoint) or a router is seen with hello packets or Database Description packets with DC bit set.

When two directly connected routers see its neighbor hello's with DC bit set means both endpoints agree to treat that link as a demand-circuit and hello's will be suppressed once the neighbor state reaches to FULL state.

Receiving a Hello with the DC-bit clear and also listing the router's Router ID in the body of the Hello message, or a Database Description Packet with the DC-bit clear (either one indicating bidirectional connectivity) indicates that the other end refuses to suppress Hellos. In these latter cases, the router reverts to the normal periodic sending of Hello Packets out the interface and the router should continue setting DC bit in its Hello and DB description packets hoping that neighbor router will come back with software capability.

LSA suppression is also performed on the interface where demand-circuit is configured.

Note: On a broadcast interface if demand circuit is enabled, Hello’s will not be suppressed but LSAs will always be suppressed.

Virtual link as demand circuit:-

Virtual links are always treated as demand circuits. In particular, over virtual links a router always negotiates to suppress the sending of Hellos and flooding LSA's.

These commands display if hellos and LSA have been suppressed.

- `show ip ospf neighbor`
- `show ip ospf neighbor detail`
- `show ip ospf interface`
- `show ip ospf virtual-links`
- `show ip ospf database`
- `show ipv6 ospf neighbor`
- `show ipv6 ospf neighbor detail`
- `show ipv6 ospf interface`
- `show ipv6 ospf virtual-links`
- `show ipv6 ospf database`

CHAPTER 11 Data Structures and Constants

This chapter describes the data structures and constants that are referred to in OSPF functions.

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`
- `prefix_ipv4`
- `prefix_ipv6`

ospf

This data structure in `ospfd/ospfd.h` defines an OSPF instance.

Member	Description
`ospf_id`	OSPF process ID
`disc_time`	OSPF discontinuity time
`start_time`	OSPF start time
`trap_flag`	SNMP trap flag
`restart_time`	Restart time
`om`	Pointer to OSPF master
`ov`	Pointer to VRF binding
`router_id`	OSPF Router ID
`router_id_config`	Router-ID configured
`domainid_list`	Pointer to OSPF Domain ID
`pdomain_id`	Pointer to configured primary domain ID
`incr_defer`	Increment defer
`tot_incr`	Total increment
`flags`	Administrative flags
`config`	Configuration variables

Member	Description
log_adj_flag	Log adjacency debug flag
abr_type	ABR type
default_origin	Default information originate
spf_start_delay	SPF timer configuration: initial SPF delay time
spf_min_delay	SPF timer configuration: minimum delay time
spf_max_delay	SPF timer configuration: maximum delay time
lsa_throttle_start	LSA throttling configuration
lsa_throttle_hold	LSA throttling configuration
lsa_throttle_max	LSA throttling configuration
min_ls_arrival	LSA throttling configuration
ref_bandwidth	Reference bandwidth: Kbps
max_dd	Maximum concurrent DD
restart_method	Method for graceful restart
restart_exit_reason	Restart exit reason
networks	Pointer to network area tables
summary	Pointer to address range for external-LSAs
nbr_static	Pointer to static neighbor for NBMA
passive_if	Pointer to passive interfaces
no_passive_if	Pointer to no passive interfaces
passive_if_default	Track default option: passive interface is set or not set
redist	Redistribute configuration
dist_in	Redistribute configuration
default_metric	Redistribute default metric
redist_update	Redistribute timer argument
area_table	Pointer to area table
if_table	Pointer to interface table
nexthop_table	Pointer to nexthop table
redist_table	Pointer to redistribute map table

Member	Description
vlink_table	Pointer to virtual interface table
multi_area_link_table	Pointer to multi area interface table
backbone	Pointer to the Backbone Area
lsdb	Pointer to LSDB of AS-external-LSAs
cspf	Pointer to CSPF instance
ext_lsdb_limit	Database overflow
exit_overflow_interval	Database overflow interval
lsdb_overflow_limit	Limit of number of LSAs
lsdb_overflow_limit_type	Soft or hard limit
max_area_limit	OSPF maximum area limit
rt_asbr	Pointer to ASBR routing table
rt_network	Pointer to IP routing table
rt_next	Pointer to candidate for nextroute calculation
t_ase_inc_calc	Pointer to ASE incremental defer timer
t_ase_calc	Pointer to ASE calculation timer
t_redist	Pointer to redistribute timer
t_lsa_event	Pointer to LS refresh event timer
t_lsa_walker	Pointer to LS refresh periodic timer
t_lsa_originate	Pointer to LSA originate event
t_nssa_inc_calc	Pointer to NSSA incremental defer timer
t_restart_state	Pointer to restart State check timer
t_grace_ack_event	Pointer to Grace LSA Ack timer
t_overflow_exit	Pointer to DB overflow exit timer
t_lsdb_overflow_event	Pointer to LSDB overflow shut down event
t_gc	Pointer to garbage collector
t_read	Pointer to read function
t_write	Pointer to write function
ibuf	Pointer to incoming buffer for receiving packets

Member	Description
obuf	Pointer to buffer for sending packets
lbuf	Pointer to buffer for LSAs
op_unuse_head	Pointer to packet unused list head
op_unuse_tail	Pointer to packet unused list tail
op_unuse_count	Pointer to packet unused count
op_unuse_max	Pointer to packet unused maximum
lsa_unuse_head	Pointer to LSA unused list head
lsa_unuse_tail	Pointer to LSA unused list tail
lsa_unuse_count	Pointer to LSA unused count
lsa_unuse_max	LSA unused maximum number
int_fd	OSPF socket for read/write
oi_write_q	Pointer to interface list of queue
distance_table	Pointer to distance parameter table
distance_all	Distance parameter
distance_inter	Distance parameter
distance_inter	Distance parameter
distance_external	Distance parameter
lsa_maxage_interval	MaxAge timer value
lsa_refresh	LSA refresh walker (struct)
lsa_event	LSA event vector
lsa_originate	LSA event vector
tv_redist	Time stamp
jitter_seed	OSPF hello timer jitter seed
dd_nbr_head	Pointer to head: concurrent DD handling
dd_nbr_tail	Pointer to tail: concurrent DD handling
dd_count_in	Incoming DD neighbors
dd_count_out	Outgoing DD neighbors
nssa_count	Counter, NSSA attached

Member	Description
incr_n_defer	Counter, NSSA incremental defer
tot_n_incr	Counter, NSSA incremental defer count
lsa_originate_count	Statistics, LSA origination
rx_lsa_count	Statistics, LSA used for new instantiation
stub_table	Pointer to OSPF stub area table
lsdb_table	Pointer to OSPF LSDB upper table
host_table	Pointer to OSPF host table
area_range_table	Pointer to OSPF area range table
nbr_table	Pointer to OSPF neighbor table
t_stamp_satistics	Cleared time stamp
t_stamp_traffic	Cleared time stamp
traffic_stats	Statistics
statistics	Statistics
t_route_cleanup	Route cleanup pointer
enable_frr	Whether loop-free alternate fast rerouting is enabled or disabled
frr_pref	Loop-free alternate fast rerouting user preferences; lower index in the array is higher priority
rt_lfa_network	Loop-free alternate routes

Definition

```
/* OSPF instance structure. */
struct ospf
{
  /* OSPF process ID. */
  u_int16_t ospf_id;

  /* OSPF discontinuity time. */

  u_int32_t disc_time;
  /* OSPF start time. */
  pal_time_t start_time;

#ifdef HAVE_SNMP
  u_int32_t trap_flag;
#endif /* HAVE_SNMP */
```

```
#ifdef HAVE_RESTART
  u_int32_t restart_time;
#endif /* HAVE_RESTART.*/

  /* Pointer to OSPF master. */
  struct ospf_master *om;

  /* VRF vinding. */
  struct ospf_vrf *ov;

  /* OSPF Router ID. */
  struct pal_in4_addr router_id;                /* OSPF Router-ID. */
  struct pal_in4_addr router_id_config;         /* Router-ID configured. */

#ifdef HAVE_VRF_OSPF
  /* OSPF Domain ID */
  struct list *domainid_list;

  /* primary domain ID */
  struct ospf_vrf_domain_id *pdomain_id;       /* Primary domain-id configured. */
#endif /* HAVE_VRF_OSPF */

  bool_t incr_defer;
  int tot_incr;
  /* Administrative flags. */
  u_int16_t flags;
#define OSPF_PROC_UP                    (1 << 0)
#define OSPF_PROC_DESTROY               (1 << 1)
#define OSPF_ROUTER_ABR                 (1 << 2)
#define OSPF_ROUTER_ASBR                (1 << 3)
#define OSPF_ROUTER_DB_OVERFLOW         (1 << 4)
#define OSPF_ROUTER_RESTART             (1 << 5)
#define OSPF_LSDB_EXCEED_OVERFLOW_LIMIT (1 << 6)
#define OSPF_ASE_CALC_SUSPENDED         (1 << 7)
#define OSPF_GRACE_LSA_ACK_RECD         (1 << 8)

#ifdef HAVE_OPAQUE_LSA
#define OSPF_AS_OPAQUE_LSA_ORIGINATOR   (1 << 9)
#endif /*HAVE_OPAQUE_LSA*/

#define OSPF_DC_BIT_CLEAR_LSA_RECD      (1 << 10)
#define OSPF_DNA_LSA_PRESENT            (1 << 11)
#define INDICATION_LSA_GEN_ALL          (1 << 12)
#define OSPF_I_LSA_PRESENCE             (1 << 13)

  /* Configuration variables. */
  u_int32_t config;
#define OSPF_CONFIG_ROUTER_ID           (1 << 0)
#define OSPF_CONFIG_DEFAULT_METRIC      (1 << 1)
#define OSPF_CONFIG_MAX_CONCURRENT_DD   (1 << 2)
```



```
#define OSPF_CONFIG_RFC1583_COMPATIBLE  (1 << 3)
#define OSPF_CONFIG_OPAQUE_LSA          (1 << 4)
#define OSPF_CONFIG_TRAFFIC_ENGINEERING (1 << 5)
#define OSPF_CONFIG_RESTART_METHOD      (1 << 6)
#define OSPF_CONFIG_OVERFLOW_LSDB_LIMIT (1 << 7)
#define OSPF_CONFIG_ROUTER_ID_USE       (1 << 8)
#ifdef HAVE_VRF_OSPF
#define OSPF_CONFIG_DOMAIN_ID_SEC       (1 << 9)
#define OSPF_CONFIG_DOMAIN_ID_PRIMARY   (1 << 10)
#define OSPF_CONFIG_NULL_DOMAIN_ID      (1 << 11)
#endif /*HAVVE_VRF_OSPF */
#define OSPF_CONFIG_DB_SUMMARY_OPT      (1 << 12)
#ifdef HAVE_BFD
#define OSPF_CONFIG_BFD                 (1 << 13)
#endif /* HAVE_BFD */
#define OSPF_CONFIG_PROTO_SHUT          (1 << 14)
#define OSPF_CONFIG_REF_BAND_IN_GBPS    (1 << 15)
#define OSPF_CONFIG_REF_BAND_IN_MBPS    (1 << 16)
#define OSPF_CONFIG_SETTRAP             (1 << 17)
#ifdef HAVE_VRF_OSPF
#define OSPF_CONFIG_VRF_LITE            (1 << 18)
#endif /* HAVE_VRF_OSPF */
#define OSPF_CONFIG_FLOOD_REDUCTION     (1 << 19)

#define OSPF_AUTO_COST_REF_BANDWIDTH_DEFAULT  0
#define OSPF_AUTO_COST_REF_BANDWIDTH_MBPS     1
#define OSPF_AUTO_COST_REF_BANDWIDTH_GBPS     2

  /* log adjacency debug flag */
  u_char log_adj_flag;
#define OSPF_LOG_ADJACENCY              (1 << 0)
#define OSPF_LOG_ADJACENCY_DETAIL       (1 << 1)

  /* ABR type. */
  u_char abr_type;

  /* Default information originate. */
  u_char default_origin;
#define OSPF_DEFAULT_ORIGINATE_UNSPEC   0
#define OSPF_DEFAULT_ORIGINATE_NSM      1
#define OSPF_DEFAULT_ORIGINATE_ALWAYS   2

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

#define OSPF_SPF_INCREMENT_VALUE        5

  /* LSA Throttling Config. */
```

```
  struct pal_timeval lsa_throttle_start;
  struct pal_timeval lsa_throttle_hold;
  struct pal_timeval lsa_throttle_max;
  struct pal_timeval min_ls_arrival;

#define OSPF_LSA_THROTTLE_INCREMENT_VALUE        2
  /* Reference bandwidth (Kbps).  */
  u_int32_t ref_bandwidth;

  /* Max concurrent DD. */
  u_int16_t max_dd;                     /* Maximum concurrent DD.  */

#ifdef HAVE_RESTART
  u_char restart_method;
#define OSPF_RESTART_METHOD_NEVER       0
#define OSPF_RESTART_METHOD_GRACEFUL    1
#define OSPF_RESTART_METHOD_SIGNALING   2
#define OSPF_RESTART_METHOD_DEFAULT     OSPF_RESTART_METHOD_GRACEFUL
#define OSPF_RESTART_GRACE_ACK_TIME     2
 u_char restart_exit_reason;
#define OSPF_RESTART_EXIT_REASON_NONE        1
#define OSPF_RESTART_EXIT_REASON_INPROGRESS  2
#define OSPF_RESTART_EXIT_REASON_COMPLETED   3
#define OSPF_RESTART_EXIT_REASON_TIMEDOUT    4
#define OSPF_RESTART_EXIT_REASON_TOPOLOGYCHG 5

#endif /* HAVE_RESTART */

#define OSPF_MAX_CONCURRENT_DD_DEFAULT  64

  /* Configuration tables. */
  struct ls_table *networks;            /* Network area tables. */
  struct ls_table *summary;             /* Address range for external-LSAs. */
  struct ls_table *nbr_static;          /* Static neighbor for NBMA. */
  struct list *passive_if;              /* Pasive interfaces. */
  struct list *no_passive_if;           /* No passive interfaces */

  /* Variable to track default option for  passive interface is set or not. */
  bool_t passive_if_default;

  /* Redistriribute configuration. */
  struct ospf_redist_conf redist[IPI_ROUTE_MAX];
  struct ospf_redist_conf dist_in;

  /* Redistribute default metric. */
  u_int32_t default_metric;
  /* Redistribute timer argument. */
  int redist_update;

  /* OSPF specific object tables. */
```

```
  struct ls_table *area_table;          /* Area table. */
  struct ls_table *if_table;            /* Interface table. */
  struct ls_table *nexthop_table;       /* Nexthop table. */
  struct ls_table *redist_table;        /* Redistribute map table. */
  struct ls_table *vlink_table;         /* Virtual interface table. */
#ifdef HAVE_OSPF_MULTI_AREA
  struct ls_table *multi_area_link_table; /* Multi area interface table. */
#endif /* HAVE_OSPF_MULTI_AREA */

  /* Pointer to the Backbone Area. */
  struct ospf_area *backbone;

  /* LSDB of AS-external-LSAs. */
  struct ospf_lsdb *lsdb;

#ifdef HAVE_OSPF_CSPF
  /* CSPF instance.  */
  struct cspf *cspf;
#endif /* HAVE_OSPF_CSPF */

#ifdef HAVE_OSPF_DB_OVERFLOW
  /* Database overflow stuff */
  int ext_lsdb_limit;
#define OSPF_DEFAULT_LSDB_LIMIT                 -1

  u_int32_t exit_overflow_interval;
#define OSPF_DEFAULT_EXIT_OVERFLOW_INTERVAL     0

#define IS_DB_OVERFLOW(O)       CHECK_FLAG((O)->flags, OSPF_ROUTER_DB_OVERFLOW)
#endif /* HAVE_OSPF_DB_OVERFLOW */

  /* Limit of number of LSAs */
  u_int32_t lsdb_overflow_limit;
  int lsdb_overflow_limit_type;         /* Soft or hard limit. */
#define OSPF_LSDB_OVERFLOW_LIMIT_OSCILATE_RANGE         0.95
#define OSPF_LSDB_OVERFLOW_SOFT_LIMIT   1
#define OSPF_LSDB_OVERFLOW_HARD_LIMIT   2
#define OSPF_AREA_LIMIT_DEFAULT                 (~0 - 1)
  u_int32_t max_area_limit;

  /* Routing tables. */
  struct ls_table *rt_asbr;             /* ASBR routing table. */
  struct ls_table *rt_network;          /* IP routing table. */
  struct ls_node *rt_next;              /* candidate for Nextroute calculation*/



  /* Threads. */
  struct thread *t_ase_inc_calc;         /* ASE incremental defer timer */
  struct thread *t_ase_calc;            /* ASE calculation timer. */
```

```
  struct thread *t_redist;              /* Redistribute timer. */
  struct thread *t_lsa_event;           /* LS refresh event timer. */
  struct thread *t_lsa_walker;          /* LS refresh pereodic timer. */
  struct thread *t_dna_lsa_walker;      /* DNA LSA Maxage Walker timer. */
  struct thread *t_dna_lsa_reach_walker;/* DNA LSA Walker timer. */
  struct thread *t_lsa_originate;       /* LSA originate event.  */
#ifdef HAVE_NSSA
  struct thread *t_nssa_inc_calc;       /* NSSA incremental defer timer */
#endif /* HAVE_NSSA */
#ifdef HAVE_RESTART
  struct thread *t_restart_state;       /* Restart State check timer. */
  struct thread *t_grace_ack_event;     /* Grace LSA Ack timer. */
#endif /* HAVE_RESTART */
#ifdef HAVE_OSPF_DB_OVERFLOW
  struct thread *t_overflow_exit;       /* DB overflow exit timer. */
#endif /* HAVE_OSPF_DB_OVERFLOW */
  struct thread *t_lsdb_overflow_event; /* LSDB overflow shut down event. */
  struct thread *t_gc;                  /* Gargabe collector. */
  struct thread *t_read;
  struct thread *t_write;

  /* Buffers and unuse list. */
  struct stream *ibuf;                  /* Buffer for receiving packets. */
  struct stream *obuf;                  /* Buffer for sending packets. */
  struct stream *lbuf;                  /* Buffer for building LSAs. */

  /* Pakcet unused list.  */
  struct ospf_packet *op_unuse_head;    /* Packet unused list head.  */
  struct ospf_packet *op_unuse_tail;    /* Packet unused list tail.  */
  u_int16_t op_unuse_count;             /* Pakcet unused count.  */
  u_int16_t op_unuse_max;               /* Packet unused maximum number.  */
#define OSPF_PACKET_UNUSE_MAX_DEFAULT   200

  /* LSA unused list.  */
  struct ospf_lsa *lsa_unuse_head;      /* LSA unused list head.  */
  struct ospf_lsa *lsa_unuse_tail;      /* LSA unused list tail.  */
  u_int16_t lsa_unuse_count;            /* LSA unused count.  */
  u_int16_t lsa_unuse_max;              /* LSA unused maximum number.  */
#define OSPF_LSA_UNUSE_MAX_DEFAULT      200

  /* OSPF socket for write. */
  int fd;

  /* Interface list of queue. */
  struct list *oi_write_q;

  /* Distance parameter. */
  struct ls_table *distance_table;
  u_char distance_all;
  u_char distance_intra;
```



```
u_char distance_inter;
u_char distance_external;

/* MaxAge timer value. */
u_int16_t lsa_maxage_interval;

/* LSA refresh walker. */
struct
{
  /* Current index. */
  u_char index;

  /* Number of rows. */
  u_char rows;

  /* Refresh interval. */
  u_int16_t interval;

  /* Refresh queue vector. */
  vector vec;
} lsa_refresh;

/* LSA DNA MAX-AGE walker. */
struct
{
  /* Current index. */
  u_char index;

  /* Number of rows. */
  u_char rows;

  /* Refresh interval. */
  u_int16_t interval;

  /* Refresh queue vector. */
  vector vec;

} dna_lsa_age;


/* LSA event vectors.  */
vector lsa_event;
vector lsa_originate;

/* Time stamp. */
struct pal_timeval tv_redist;

/* OSPF hello timer jitter seed.  */
u_int32_t jitter_seed;
```

```
  /* Concurrent DD handling.  */
  struct ospf_neighbor *dd_nbr_head;
  struct ospf_neighbor *dd_nbr_tail;
  u_int16_t dd_count_in;               /* Incomming DD neighbors.  */
  u_int16_t dd_count_out;              /* Outgoing DD neighbors.  */

  /* Counters.  */
#ifdef HAVE_NSSA
  u_int32_t nssa_count;                /* NSSA attached. */
  bool_t incr_n_defer;                 /* NSSA incremental defer */
  int tot_n_incr;                      /* NSSA incremental defer count */
#endif /* HAVE_NSSA */

  /* Statistics. */
  u_int32_t lsa_originate_count;       /* LSA origination. */
  u_int32_t rx_lsa_count;              /* LSA used for new instantiation. */

  /* API related tables. */
  struct ls_table *stub_table;         /* OSPF stub area table. */
  struct ls_table *lsdb_table;         /* OSPF LSDB upper table. */
  struct ls_table *host_table;         /* OSPF Host table. */
  struct ls_table *area_range_table;   /* OSPF Area Range table. */
  struct ls_table *nbr_table;          /* OSPF Neighbor table. */

  /* Current time for avoiding system call many times.  */
  //struct pal_timeval tv_current;

  /* Cleared time stamp */
  pal_time_t t_stamp_satistics;
  pal_time_t t_stamp_traffic;

  struct
  {
    /* Statistics. */
    u_int32_t hello_in;          /* Hello packet input count. */
    u_int32_t hello_out;         /* Hello packet output count. */
    u_int32_t db_desc_in;        /* database desc. packet input count. */
    u_int32_t db_desc_out;       /* database desc. packet output count. */
    u_int32_t ls_req_in;         /* LS request packet input count. */
    u_int32_t ls_req_out;        /* LS request packet output count. */
    u_int32_t ls_upd_in;         /* LS update packet input count. */
    u_int32_t ls_upd_out;        /* LS update packet output count. */
    u_int32_t ls_ack_in;         /* LS Ack packet input count. */
    u_int32_t ls_ack_out;        /* LS Ack packet output count. */
    /* Error counters */
    u_int32_t discarded;         /* Discarded input count by error. */
    u_int32_t discard_out;
    u_int32_t hellosin;
    u_int32_t dbsin;
    u_int32_t lsreqin;
```



```
    u_int32_t lsackin;
    u_int32_t unknown_in;
    u_int32_t unknown_out;
    u_int32_t badcrc;
    u_int32_t wrong_area;
    u_int32_t bad_version;
    u_int32_t bad_auth;
    u_int32_t passive;
    u_int32_t nonbr;
    u_int32_t invalid_src;
    u_int32_t invalid_dst;
    u_int32_t pktlength;

  } traffic_stats;

  struct
  {
    #define OSPF_LSA_STATS_ORIGINATED      0
    #define OSPF_LSA_STATS_REFRESH         1
    #define OSPF_LSA_STATS_FLUSHED         2
    #define OSPF_LSA_STATS_AGED_OUT        3
    #define OSPF_LSA_STATS_MAX             4

    u_int32_t router_id_change_counter;
    u_int32_t dr_election_counter;
    u_int32_t older_lsas_counter;
    u_int32_t nbr_state_change_counter;
    u_int32_t nbr_bad_lsreqs_counter;
    u_int32_t nbr_interval_expired_counter;
    u_int32_t nbr_seq_number_mismatch;
    u_int32_t spf_full;
    u_int32_t spf_summary;
    u_int32_t spf_external;
    u_int32_t lsa_counters[OSPF_LSA_STATS_MAX][OSPF_MAX_LSA];
  } statistics;

  struct thread *t_route_cleanup;
#ifdef HAVE_OSPF_LFA

  /* flag to indicate whether lfa/ip frr is enable or not */
  bool_t enable_frr;

  /* Keeps user preferences, lower index in the array is higher priority */
#define FRR_PREF_MAX 10
  u_int8_t frr_pref[FRR_PREF_MAX];

  /* data structure to keep lfa routes */
  struct ls_table *rt_lfa_network;

#endif /* HAVE_OSPF_LFA */
```

```
#if defined(HAVE_HA) && !defined(HAVE_HA_OSPF_GR)
  /* CDR refrecnce for ospf */
  HA_CDR_REF ospf_instance_cdr_ref;
#endif /* HAVE_HA && !HAVE_HA_OSPF_GR */
};
```

ospf_master

This data structure in `ospfd/ospfd.h` is for system-wide configuration and variables.

.

Member	Description
`vr`	Pointer to virtual router
`zg`	Pointer to library globals
`ospf`	List of ospf instances
`cspf`	List of cspf instances
`debug_cspf`	CSPF debug flag
`conf`	Debug flag for configuration
`term`	Debug flag for terminal
`vlink_index`	Virtual Interface Index
`flags`	OSPF global flags
`config`	OSPF global configuration
`restart_reason`	Restart reason for Graceful restart
`helper_policy`	Helper policy for Graceful Restart
`max_grace_period`	Helper limit grace period for Graceful Restart
`grace_period`	Grace period for Graceful Restart
`restart_option`	Buffer for restart option
`restart_tlvs`	Buffer for restart option
`t_restart`	Pointer to Graceful Restart expire timer/enter event
`t_ls_upd_proc`	Thread for delayed processing of LS Updates
`t_ls_upd_send_event`	Thread for flooding LSAs
`never_restart_helper_list`	OSPF Helper Never Router-ID list

Member	Description
if_table	OSPF global interface table
if_params	OSPF interface parameter pool
notifiers	Pointer to OSPF notifier
trap	SNMP trap callback function

Definition

```
/* OSPF master for system wide configurations and variables. */
struct ospf_master
{
  /* Pointer to VR. */
  struct ipi_vr *vr;

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

  /* OSPF instance list. */
  struct list *ospf;

#ifdef HAVE_OSPF_CSPF
  /* CSPF instance. */
  struct cspf *cspf;
  struct cspf_debug_flags debug_cspf;
#endif /* HAVE_OSPF_CSPF */

  /* OSPF debug flags. */
  struct
  {
    /* Debug flags for configuration. */
    struct debug_ospf conf;

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

  } debug;

  /* Virtual interface index. */
  int vlink_index;

  /* OSPF global flags. */
  u_char flags;
#define OSPF_GLOBAL_FLAG_RESTART                        (1 << 0)
#ifdef HAVE_OSPF_MULTI_INST
#define OSPF_EXT_MULTI_INST                             (1 << 1)
#endif /* HAVE_OSPF_MULTI_INST */
```

```
#ifdef HAVE_RESTART
  /* OSPF global config. */
  u_char config;
#define OSPF_GLOBAL_CONFIG_RESTART_GRACE_PERIOD         (1 << 0)
#define OSPF_GLOBAL_CONFIG_RESTART_HELPER_POLICY        (1 << 1)
#define OSPF_GLOBAL_CONFIG_RESTART_HELPER_GRACE_PERIOD  (1 << 2)
#define OSPF_GLOBAL_CONFIG_RESTART_PLANNED_ONLY         (1 << 3)

  /* Restart reason for Graceful Restart. */
  u_char restart_reason;
#define OSPF_RESTART_REASON_UNKNOWN                      0
#define OSPF_RESTART_REASON_RESTART                      1
#define OSPF_RESTART_REASON_UPGRADE                      2
#define OSPF_RESTART_REASON_SWITCH_REDUNDANT             3

  /* Helper policy for Graceful Restart. */
  u_char helper_policy;
#define OSPF_RESTART_HELPER_UNSPEC                       0
#define OSPF_RESTART_HELPER_NEVER                        1
#define OSPF_RESTART_HELPER_ONLY_RELOAD                  2
#define OSPF_RESTART_HELPER_ONLY_UPGRADE                 3

  /* Helper limit grace period for Graceful Restart. */
  u_int16_t max_grace_period;

  /* Grace period for Graceful Restart. */
  u_int16_t grace_period;

  /* Buffer for restart option. */
  struct stream *restart_option;
  vector restart_tlvs;

  /* Graceful Restart expire timer/enter event. */
  struct thread *t_restart;

  /* OSPF Helper Never Router-ID list */
  struct list *never_restart_helper_list;
#endif /* HAVE_RESTART */

  /* Thread for delayed processing of LS Updates */
  struct thread *t_ls_upd_proc;

  /* Thread for flooding of LSAs */
  struct thread *t_ls_upd_send_event;

  /* OSPF global interface table. */
  struct ls_table *if_table;

  /* OSPF interface parameter pool. */
  struct list *if_params;
```



```
  /* OSPF notifiers. */
  struct list *notifiers[OSPF_NOTIFY_MAX];

  /* List of ospf interfaces, where LSA
     needs to be flooded */
  struct list *ls_upd_oi_list;
  struct listnode *curr_oi;

  /* List of Neighbors, from which LS Updates
     needs to be processed */
  struct list *ls_upd_nbr_list;

  /* OSPF SNMP trap callback function */
#ifdef HAVE_SNMP
  vector traps[OSPF_TRAP_ID_MAX];
#endif /* HAVE_SNMP */

#if defined(HAVE_HA) && !defined(HAVE_HA_OSPF_GR)
  /* Flag indicating the recovery status */
  bool_t  is_recovering;
  /* CDR refrecnce for ospf_master */
  HA_CDR_REF ospf_master_cdr_ref;
#endif /* HAVE_HA && !HAVE_HA_OSPF_GR */
};
```

ospf6

This data structure in `ospf6d/ospf6d.h` is for system-wide configuration and variables.

Definition

```
struct ospf6
{
  /* OSPFv3 process tag. */
  char *tag;

  /* OSPFv3 start time. */
  pal_time_t start_time;

  /* OSPFv3 discontinuity time. */
  u_int32_t disc_time;

#ifdef HAVE_RESTART
  u_int32_t restart_time;
#endif /* HAVE_RESTART.*/

  /* Pointer to OSPFv3 master.  */
  struct ospf6_master *om;
```

```
  /* VRF vinding. */
  struct ospf6_vrf *ov;

  /* OSPFv3 Router ID. */
  struct pal_in4_addr router_id;
  struct pal_in4_addr router_id_config;         /* Router-ID configured. */

  /* OSPFv3 Address Family. */
  u_char add_family;

  /* Administrative flags. */
  u_int16_t flags;
#define OSPF_PROC_UP                    (1 << 0)
#define OSPF_PROC_DESTROY               (1 << 1)
#define OSPF_ROUTER_ABR                 (1 << 2)
#define OSPF_ROUTER_ASBR                (1 << 3)
#define OSPF_ASE_CALC_SUSPENDED         (1 << 4)
#define OSPF_ROUTER_RESTART             (1 << 5)
#define OSPF_ROUTER_HELPER              (1 << 6)
#define OSPF_GRACE_LSA_ACK_RECD         (1 << 7)
#define OSPF6_DC_BIT_CLEAR_LSA_RECD     (1 << 8)
#define OSPF6_DNA_LSA_PRESENT           (1 << 9)
#define INDICATION_LSA_GEN_ALL          (1 << 10)
  /* Configuration variables. */
  u_int16_t config;
#define OSPF6_CONFIG_TE_ENABLED                 (1 << 0)
#define OSPF6_CONFIG_MAX_CONCURRENT_DD          (1 << 1)
#define OSPF6_CONFIG_ROUTER_ID_USE              (1 << 2)
#define OSPF6_CONFIG_ROUTER_ID                  (1 << 3)
#define OSPF6_CONFIG_RESTART_METHOD             (1 << 4)
#define OSPF6_CONFIG_DB_SUMMARY_OPT             (1 << 5)
#define OSPF6_CONFIG_DEFAULT_METRIC             (1 << 6)
#ifdef HAVE_BFD
#define OSPF6_CONFIG_BFD                        (1 << 7)
#endif /* HAVE_BFD */
#define OSPF6_CONFIG_LOG_ADJACENCY_CHANGES      (1 << 8)
#define OSPF6_CONFIG_LOG_ADJACENCY_DETAIL       (1 << 9)
#define OSPF6_CONFIG_PROC_SHUT                  (1 << 10)
#define OSPF6_CONFIG_REF_BW_IN_GBPS             (1 << 11)
#define OSPF6_CONFIG_REF_BW_IN_MBPS             (1 << 12)
#ifdef HAVE_OSPF6_LFA
#define OSPF6_CONFIG_FRR                        (1 << 13)
#endif /*HAVE_OSPF6_LFA*/
#ifdef HAVE_OSPF6_OD
#define OSPF6_CONFIG_FLOOD_REDUCTION            (1 << 14)
#endif /* HAVE_OSPF6_OD */
  /* ABR type. */
  u_char abr_type;
```

```
  /* Default information originate. */
  u_char default_origin;

#define OSPF6_DEFAULT_ORIGINATE_UNSPEC   0
#define OSPF6_DEFAULT_ORIGINATE_NSM      1
#define OSPF6_DEFAULT_ORIGINATE_ALWAYS   2

  /* Pointer to the Backbone Area. */
  struct ospf6_area *backbone;

  /* LSDB of AS-external-LSAs. */
  struct ospf6_lsdb *lsdb;

#ifdef HAVE_OSPF6_CSPF
  /* CSPF instance.  */
  struct cspf *cspf;
#endif /* HAVE_OSPF6_CSPF */

  /* Tables. */
  struct ls_table *area_table;          /* Area table. */
  struct ls_table *if_table;            /* Interface table. */
  struct ls_table *nexthop_table;       /* Nexhtop table. */
  struct ls_table *redist_table;        /* Redistribute map table. */
  struct ls_table *vlink_table;         /* Virtual-Link table. */
  struct ls_table *summary;             /* Address range for external-LSAs. */

  /* Routing tables. */
  struct ls_table *rt_asbr;             /* ASBR routing table. */
  struct ls_table *rt_network;          /* IPv6 network routing table. */

  /* SPF timer config. */
  struct pal_timeval spf_start_delay;        /* Initial SPF delay time. */
  struct pal_timeval spf_min_delay;         /* SPF minimum delay time. */
  struct pal_timeval spf_max_delay;         /* SPF maximum delay time. */
#define OSPF6_SPF_INCREMENT_VALUE        5


  /* Reference bandwidth (Kbps).  */
  u_int32_t ref_bandwidth;
#define OSPF6_REF_BW_TYPE_DEFAULT        0
#define OSPF6_REF_BW_TYPE_GBPS           1
#define OSPF6_REF_BW_TYPE_MBPS           2
  /* Max concurrent DD. */
  u_int16_t max_dd;                     /* Maximum concurrent DD.  */
#define OSPF6_MAX_CONCURRENT_DD_DEFAULT 5

#ifdef HAVE_RESTART
  u_char restart_method;
#define OSPF6_RESTART_METHOD_NEVER       0
#define OSPF6_RESTART_METHOD_GRACEFUL    1
```

```
#define OSPF6_RESTART_METHOD_DEFAULT     OSPF6_RESTART_METHOD_GRACEFUL
#define OSPF6_RESTART_GRACE_ACK_TIME     2
  u_char restart_exit_reason;
#define OSPF6_RESTART_EXIT_REASON_NONE        1
#define OSPF6_RESTART_EXIT_REASON_INPROGRESS  2
#define OSPF6_RESTART_EXIT_REASON_COMPLETED   3
#define OSPF6_RESTART_EXIT_REASON_TIMEDOUT    4
#define OSPF6_RESTART_EXIT_REASON_TOPOLOGYCHG 5
#endif /* HAVE_RESTART */

  /* 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 */

  /* Configuration redistribute. */
  struct ospf6_redist_conf redist[IPI_ROUTE_MAX];

  struct ospf6_redist_conf dist_in;

  /* Default metric for redistribution. */
  u_int32_t default_metric;

  /* Redistribute timer argument. */
  int redist_update;

  /* Buffers and unuse list.  */
  struct stream *ibuf;                  /* Buffer for receiving packets.  */
  struct stream *obuf;                  /* Buffer for sending packets.  */
  struct stream *lbuf;                  /* Buffer for building LSAs. */

  /* Packet unuse list.  */
  struct ospf6_packet *op_unuse_head;   /* Packet unused list head.  */
  struct ospf6_packet *op_unuse_tail;   /* Packet unused list tail.  */
  u_int32_t op_unuse_count;             /* Packet unused count.  */
  u_int32_t op_unuse_max;               /* Packet unused maximum number.  */
#define OSPF6_PACKET_UNUSE_MAX_DEFAULT  200

  /* LSA unused list.  */
  struct ospf6_lsa *lsa_unuse_head;     /* LSA unused list head.  */
  struct ospf6_lsa *lsa_unuse_tail;     /* LSA unused list tail.  */
  u_int32_t lsa_unuse_count;            /* LSA unused count.  */
  u_int32_t lsa_unuse_max;              /* LSA unused maximum number.  */
#define OSPF6_LSA_UNUSE_MAX_DEFAULT     200

  /* Socket file descriptor. */
  int fd;

  /* Distance parameter. */
  u_char distance_all;
```



```
  u_char distance_intra;
  u_char distance_inter;
  u_char distance_external;


  /* Write queue. */
  struct list *oi_write_q;

  /* Threads. */
  struct thread *t_read;               /* Read thread.  */
  struct thread *t_write;              /* Write thread.  */
  struct thread *t_redist;             /* Redistribute timer.  */
  struct thread *t_ase_calc;           /* ASE calculation timer.  */
  struct thread *t_gc;                 /* Garbage collector.  */
#ifdef HAVE_RESTART
  struct thread *t_restart_state;      /* Restart State check timer. */
  struct thread *t_grace_ack_event;    /* Grace LSA Ack timer. */
#endif /* HAVE_RESTART */

##ifdef HAVE_OSPF6_LFA
  /* Keeps user preferences, lower index in the array is higher priority */
#define FRR_PREF_MAX 4
  u_int8_t frr_pref[FRR_PREF_MAX];

  /* data structure to keep lfa routes */
  struct ls_table *rt_lfa_network;

#endif /* HAVE_OSPF6_LFA */

  /* Timestamp. */
  struct pal_timeval tv_redist;

  /* OSPF hello timer jitter seed.  */
  u_int32_t jitter_seed;

  /* Concurrent DD handling.  */
  struct ospf6_neighbor *dd_nbr_head;
  struct ospf6_neighbor *dd_nbr_tail;
  u_int16_t dd_count_in;               /* Incomming DD neighbors.  */
  u_int16_t dd_count_out;              /* Outgoing DD neighbors.  */

  /* Statistics.  */
  u_int32_t lsa_self_count;         /* LSA origination. */
  u_int32_t rx_lsa_count;          /* LSA received for new instantiations. */

  /* OSPF6 Process ID */
  u_int32_t pid;

  /* Counters */
#ifdef HAVE_NSSA
```

```
  u_int32_t nssa_count;               /* NSSA attached */
#endif /* HAVE_NSSA */
};
```

ospf6_master

This data structure in `ospf6d/ospf6d.h` is for system-wide configuration and variables.

Definition

```
/* OSPFv3 master for system wide configurations and variables. */
struct ospf6_master
{
  /* Pointer to VR.  */
  struct ipi_vr *vr;

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

  /* OSPFv3 process list. */
  struct list *proclist;

#ifdef HAVE_OSPF6_CSPF
  /* CSPF instance. */
  struct cspf *cspf;
  struct cspf_debug_flags debug_cspf;
#endif /* HAVE_OSPF6_CSPF */

  /* OSPFv3 start time. */
  struct pal_timeval start_time;

  /* OSPFv3 global flags. */
  u_int32_t flags;
#define OSPF6_DISPLAY_SINGLE_LINE       (1 << 0)
#define OSPF6_GLOBAL_FLAG_RESTART       (1 << 1)

#ifdef HAVE_RESTART
  /* OSPFv3 global config. */
  u_char config;
#define OSPF6_GLOBAL_CONFIG_RESTART_GRACE_PERIOD         (1 << 0)
#define OSPF6_GLOBAL_CONFIG_RESTART_HELPER_POLICY        (1 << 1)
#define OSPF6_GLOBAL_CONFIG_RESTART_HELPER_GRACE_PERIOD  (1 << 2)
#define OSPF6_GLOBAL_CONFIG_RESTART_PLANNED_ONLY         (1 << 3)
  /* Restart reason for Graceful Restart. */
  u_char restart_reason;
#define OSPF6_RESTART_REASON_UNKNOWN                     0
#define OSPF6_RESTART_REASON_SOFTWARE                    1
#define OSPF6_RESTART_REASON_UPGRADE                     2
```

```
#define OSPF6_RESTART_REASON_RELOAD                     3
#define OSPF6_RESTART_REASON_SWITCH_TO_REDUNDANT        4

  /* Helper policy for Graceful Restart. */
  u_char helper_policy;
#define OSPF6_RESTART_HELPER_UNSPEC                     (1 << 0)
#define OSPF6_RESTART_HELPER_NEVER                      (1 << 1)
#define OSPF6_RESTART_HELPER_ONLY_RELOAD                (1 << 2)
#define OSPF6_RESTART_HELPER_ONLY_UPGRADE               (1 << 3)
#define OSPF6_RESTART_HELPER_NEVER_ROUTERID             (1 << 4)

  /* Helper limit grace period for Graceful Restart. */
  u_int16_t max_grace_period;

  /* Grace period for Graceful Restart. */
  u_int16_t grace_period;

  /* Buffer for restart option. */
  struct stream *restart_option;
  vector restart_tlvs;
  /* Graceful Restart expire timer/enter event. */
  struct thread *t_restart;

  /* OSPF Helper Never Router-id list. */
  struct list *never_restart_helper_list;
#endif /* HAVE_RESTART */

  /* OSPFv3 interface parameter list. */
  struct list *if_params;

  /* Debug flags. */
  struct
  {
    /* Debug flags for configuration. */
    struct debug_ospf6 conf;
    /* Debug flags for terminal. */
    struct debug_ospf6 term;

  } debug;

  /* Redistribute info table. */
  struct ls_table *redist_table;
  u_int32_t redist_count[AF_IPV4_MCAST + 1][IPI_ROUTE_MAX];

  /* Master interface index for virtual-interface. */
  u_int32_t vlink_index;

  /* OSPF notifiers */
  struct list *notifiers[OSPF_NOTIFY_MAX];
```

```
  /* Bitmap */
  struct bitmap *bitmap;
};
```

Protocol Constants

These protocol types are defined in `pal/api/pal_types.def`:

Constant	Description
`IPI_ROUTE_BGP`	BGP routes
`IPI_ROUTE_CONNECT`	Connected routes
`IPI_ROUTE_DEFAULT`	System
`IPI_ROUTE_ISIS`	IS-IS routes
`IPI_ROUTE_KERNEL`	Kernel routes
`IPI_ROUTE_OSPF`	OSPFv2 routes
`IPI_ROUTE_OSPF6`	OSPFv3 routes
`IPI_ROUTE_RIP`	RIP routes
`IPI_ROUTE_RIPNG`	RIPng routes
`IPI_ROUTE_STATIC`	Static routes
`IPI_ROUTE_TRILL`	TRILL routes

CHAPTER 12 OSPFv2 Command API

This chapter describes the OSPFv2 Command APIs. The following sections are covered in this section:

- OSPFv2 API
- OSPFv2 Graceful Restart API
- OSPFv2 Passive-Interface API

OSPFv2 API

The following is a summary of the OSPFv2 functions. Details are provided in the following subsections.

Function	Description
ospf_abr_type_set	Sets the ABR type
ospf_abr_type_unset	Resets the configured ABR type
ospf_area_auth_type_set	Enables the specified area authentication type
ospf_area_auth_type_unset	Disables the authentication type for the area
ospf_area_default_cost_set	Sets the default cost value of a not so stubby area or stub area
ospf_area_default_cost_unset	Resets the cost to the default value
ospf_area_export_list_set	Sets the type-3 export filter for networks announced to other areas
ospf_area_export_list_unset	Resets the export list
ospf_area_filter_list_access_set	Sets the type-3 LSA filter with the access list name
ospf_area_filter_list_access_unset	Resets the filter list access configuration
ospf_area_filter_list_prefix_set	Sets the prefix list value as the type-3 LSA filter
ospf_area_filter_list_prefix_unset	Cancels the filter prefix advertise
ospf_area_import_list_set	Sets the import list value for the type-3 import filter
ospf_area_import_list_unset	Resets the import list value for the type-3 import
ospf_area_no_summary_set	Sets the OSPF area as stub
ospf_area_no_summary_unset	Removes the no-summary restriction
ospf_area_nssa_default_originate_set	Originates a default route into a not so stubby area (NSSA)
ospf_area_nssa_default_originate_unset	Removes the default route from the not so stubby area (NSSA)
ospf_area_nssa_no_redistribution_set	Defines an area parameter for NSSA
ospf_area_nssa_no_redistribution_unset	Resets OSPF so that redistribution in not allowed to the stub

Function	Description
ospf_area_nssa_set	Defines an area as an NSSA
ospf_area_nssa_stability_interval_set	Sets the NSSA stability interval
ospf_area_nssa_translator_role_set	Sets the value of the translator parameter of the area
ospf_area_nssa_translator_role_unset	Removes the value of the translator parameter of the area
ospf_area_nssa_unset	Removes the NSSA designation from the specified area
ospf_area_range_not_advertise_set	Sets the ABR to not advertise the summary LSA for each route
ospf_area_range_not_advertise_unset	Create a summary LSA for each route in specific area; advertise in adjacent areas
ospf_area_range_set	Advertise a single route for a range of addresses
ospf_area_range_substitute_set	Substitutes summarized prefix with another prefix within the range for an area
ospf_area_range_substitute_unset	Removes summarized prefix with another prefix for an area
ospf_area_range_unset	Unsets the area range
ospf_area_shortcut_set	Sets the shortcut mode of the specified area
ospf_area_shortcut_unset	Removes the shortcut mode of the specified area
ospf_area_stub_set	Sets the specified area as a stub
ospf_area_stub_unset	Removes the stub definition from the specified area
ospf_auto_cost_reference_bandwidth_set	Sets the bandwidth value
ospf_auto_cost_reference_bandwidth_unset	Disables the bandwidth restriction
ospf_bfd_all_interfaces_set	Enables BFD on all interfaces
ospf_bfd_all_interfaces_unset	Disables BFD on all interfaces
ospf_capability_cspf_set	Sets the CSPF capability for an OSPF process
ospf_capability_cspf_unset	Disables the CSPF capability for an OSPF process
ospf_capability_opaque_lsa_set	Sets the opaque capability for an OSPF process
ospf_capability_opaque_lsa_unset	Unsets the opaque capability for an OSPF process
ospf_capability_traffic_engineering_set	Sets the traffic engineering capability for an OSPF process
ospf_capability_traffic_engineering_unset	Unsets the traffic engineering capability for an OSPF process
ospf_compatible_rfc1583_set	Calculates route summary costs according to RFC 1583
ospf_compatible_rfc1583_unset	Disables the calculation of route summary costs according to RFC 1583
ospf_default_metric_set	Sets the default metric for redistributed routes

Function	Description
ospf_default_metric_unset	Disables the default metric for the specified process
ospf_disable_db_summary_opt	Disables the OSPF Database Summary List optimization
ospf_disable_ext_multi_inst	Disables the support of multiple OSPF instances on a subnet
ospf_distance_all_set	Sets the value of administrative distance for all route types
ospf_distance_all_unset	Resets the specified value of administrative distance for all route types
ospf_distance_external_set	Sets the specified administrative distance of the external route
ospf_distance_external_unset	Resets the administrative distance value to 0
ospf_distance_inter_area_set	Sets the specified distance value for inter-area routes
ospf_distance_inter_area_unset	Resets the administrative distance value to 0
ospf_distance_intra_area_set	Sets the specified distance value of the route
ospf_distance_intra_area_unset	Resets the administrative distance value to 0
ospf_distance_source_set	Sets the administrative distance to prefixes whose nexthop matches the given source IP address
ospf_distance_source_unset	Unsets the administrative distance to prefixes whose nexthop matches the given source IP address
ospf_distribute_list_in_set	Sets the inbound distribute list for the specified protocol
ospf_distribute_list_out_unset	Unsets the inbound distribute list for the specified protocol
ospf_distribute_list_out_set	Sets the outbound distribute list for the specified protocol
ospf_distribute_list_out_unset	Unsets the outbound distribute list for the specified protocol
ospf_domain_id_set	Sets an OSPF domain ID
ospf_domain_id_unset	Unsets an OSPF domain ID
ospf_dna_set	Enables flood reduction feature on all OSPF enabled interfaces
ospf_dna_unset	Disables flood reduction feature on all OSPF enabled interface
ospf_enable_db_summary_opt	Enables the OSPF Database Summary List optimization
ospf_enable_ext_multi_inst	Enables multiple OSPF instances to run on a subnet
ospf_if_authentication_key_set	Sets the authentication key for an area
ospf_if_authentication_key_set_by_addr	Sets the authentication key for the interface specified by address
ospf_if_authentication_key_unset	Removes the authentication key for an area
ospf_if_authentication_key_unset_by_addr	Resets the authentication key for the interface specified by address
ospf_if_conf_ldp_igp_sync	Enables LDP-OSPF synchronization on the current interface

Function	Description
ospf_if_conf_ldp_igp_unsync	Disables LDP-OSPF synchronization on the current interface
ospf_if_cost_set	Sets the current interface output cost
ospf_if_cost_set_by_addr	Sets the output cost of the interface of the specific IP address
ospf_if_cost_unset	Resets the cost for the current interface
ospf_if_cost_unset_by_addr	Resets the cost for the specified interface
ospf_if_database_filter_set	Suppresses all LSA during synchronization and flooding on a particular interface
ospf_if_database_filter_set_by_addr	Suppresses all LSA during synchronization and flooding of the specified interface and IP address
ospf_if_database_filter_unset	Unconfigures the database filter of the specified interface
ospf_if_database_filter_unset_by_addr	Unconfigures the database filter of the interface specified by IP address
ospf_if_dc_set	Enables ospf demand-circuit for the interface
ospf_if_dc_unset	Disables ospf demand-circuit for the interface
ospf_if_dead_interval_set	Sets the router-dead-interval for the interface
ospf_if_dead_interval_set_by_addr	Sets the router-dead-interval for the interface of the given IP address
ospf_if_dead_interval_unset	Resets the dead interval of the specified interface to its default value
ospf_if_dead_interval_unset_by_addr	Resets the dead interval of the interface specified by IP address to its default value
ospf_if_disable_all_set	Disables all packet processing on the interface
ospf_if_disable_all_unset	Sets the hello interval for the current interface
ospf_if_dna_set	Enables flood reduction feature on an interface
ospf_if_dna_unset	Disables flood reduction feature on an interface
ospf_if_hello_interval_set	Sets the hello interval for the current interface
ospf_if_hello_interval_set_by_addr	Sets the hello interval for the interface specified by IP address
ospf_if_hello_interval_unset	Resets the hello interval of the current interface to its default value
ospf_if_hello_interval_unset_by_addr	Resets the hello interval of the interface specified by IP address to its default value
ospf_if_message_digest_key_set	Sets the MD5 key for the current interface
ospf_if_message_digest_key_set_by_addr	Sets the MD5 key for the interface specified by IP address
ospf_if_message_digest_key_unset	Resets the MD5 key for the current interface
ospf_if_message_digest_key_unset_by_addr	Resets the MD5 key for the interface specified by IP address

Function	Description
ospf_if_network_p2mp_nbma_set	Configures an interface to Point-to-Multipoint Non-Broadcast mode
ospf_if_network_set	Sets the network type as Point-to-Point, Broadcast, NBMA or Point-to-MultiPoint
ospf_if_network_unset	Resets the network type to its default type
ospf_if_priority_set	Resets the priority of the current interface
ospf_if_priority_set_by_addr	Turns on redistribute from the specified protocol
ospf_if_priority_unset	Resets the priority of the current interface
ospf_if_priority_unset_by_addr	Resets the priority of the interface specified by IP address
ospf_if_resync_timeout_set	Sets the re-synch timeout interval (seconds)
ospf_if_retransmit_interval_set	Sets the retransmit interval for the current interface
ospf_if_retransmit_interval_set_by_addr	Sets the retransmit interval for the specified interface
ospf_if_retransmit_interval_unset	Resets the retransmit interval of the current interface to its default value
ospf_if_retransmit_interval_unset_by_addr	Resets the retransmit interval of the interface specified by IP address to its default value
ospf_if_te_metric_set	Sets the TE-metric on the specified interface
ospf_if_te_metric_unset	Unsets the TE-metric on a particular interface
ospf_if_transmit_delay_set	Sets the transmit delay interval for the current interface
ospf_if_transmit_delay_set_by_addr	Sets the transmit delay interval for the interface specified by IP address
ospf_if_transmit_delay_unset	Resets the transmit delay interval of the current interface to its default value
ospf_if_transmit_delay_unset_by_addr	Resets the transmit delay interval of the interface specified by IP address to its default value
ospf_lsa_min_arrival_set	Sets the minimum interval to accept the same LSA from neighbors
ospf_lsa_min_arrival_unset	Sets the minimum interval to accept the same LSA from neighbors to its default value (1000 milliseconds)
ospf_lsa_throttle_timers_set	Sets the rate-limiting intervals for LSA generation
ospf_lsa_throttle_timers_unset	Sets the rate-limiting intervals for LSA generation to their default values
ospf_max_area_limit_set	Sets maximum number of OSPF areas
ospf_max_area_limit_unset	Resets maximum number of OSPF areas
ospf_max_concurrent_dd_set	Sets the maximum number of Database Descriptors to concurrently process
ospf_max_concurrent_dd_unset	Resets the maximum number of Database Descriptors to concurrently process
ospf_multi_area_adjacency_unset	Disables multi-area-adjacency

Function	Description
ospf_nbr_static_config_check	Checks the configuration status of a neighbor
ospf_nbr_static_cost_set	Sets the cost of the non-broadcast multi-access (NBMA) neighbor
ospf_nbr_static_cost_unset	Unconfigures the cost of the specified NBMA neighbor
ospf_nbr_static_poll_interval_set	Sets the poll interval of the NBMA neighbor
ospf_nbr_static_poll_interval_unset	Resets the poll interval of the specified NBMA neighbor to its default value
ospf_nbr_static_priority_set	Sets the priority of the NBMA neighbor
ospf_nbr_static_priority_unset	Resets the priority of the NBMA neighbor to its default value
ospf_nbr_static_set	Sets the NBMA neighbor
ospf_nbr_static_unset	Deletes the static NBMA neighbor
ospf_network_format_set	Sets the Area ID for the specified address format
ospf_network_set	Enables an interface for the OSPF domain
ospf_network_unset	Disables the interface to the OSPF domain
ospf_network_wildmask_set	Sets the mask representation for the specified network area
ospf_opaque_area_lsa_set	Generates area Opaque LSAs
ospf_opaque_as_lsa_set	Generates area Opaque LSAs
ospf_opaque_data_validate_and_send	Validates the Opaque LSA and sends the relevant information to the requesting application
ospf_opaque_link_lsa_set	Generates the specified AS-wide Opaque LSA
ospf_overflow_database_external_interval_set	Sets the value of the time-to-recover interval of the overflow state
ospf_overflow_database_external_interval_unset	Resets the interval of the overflow state to its default value
ospf_overflow_database_external_limit_set	Sets the maximum number of LSAs as specified
ospf_overflow_database_external_limit_unset	Resets the limit of overflow state to its default value
ospf_process_set	Creates an OSPF instance
ospf_process_unset	Destroys the specified OSPF process
ospf_redistribute_default_set	Turns on the default originate with the specified origin
ospf_redistribute_set	Turns on redistribute from the specified protocol
ospf_routemap_set	Sets the route-map to specified protocol
ospf_routemap_unset	resets the route-map to the default protocol
ospf_router_id_set	Sets the static OSPF router ID to the specified value
ospf_router_id_unset	Resets the static OSPF router ID to the default value (0)

Function	Description
ospf_set_frr	Enables or disables fast rerouting
ospf_set_frr_interface	Permits an interface to be used as the next hop in a repair path or prohibits an interface from being used as the next hop in a repair path
ospf_set_frr_tie_break_prority	Sets the priority for a type of fast reroute repair path
ospf_summary_address_not_advertise_set	Sets the flag of the external summary address range to NotAdvertise
ospf_summary_address_not_advertise_unset	Resets the flag of the external summary address range
ospf_summary_address_set	Sets the external summary address range
ospf_summary_address_tag_set	Sets the tag value to the specified value
ospf_summary_address_tag_unset	Resets the tag value of the external summary address range (0)
ospf_summary_address_unset	Resets the external summary address range
ospf_timers_refresh_set	Sets the LSA refresh timer
ospf_timers_refresh_unset	Resets the LSA refresh timer to default value
ospf_timers_spf_set	Rets the minimum and maximum delay between a topology change
ospf_timers_spf_unset	Resets the shortest path first (SPF) minimum and maximum delays to default values
ospf_unset_frr_tie_break_prority_all	Sets the priority for all types of fast reroute repair paths to their default values
ospf_vlink_authentication_key_set	Sets the authentication type for the virtual interface
ospf_vlink_authentication_key_unset	Resets the simple authentication password for the virtual interface
ospf_vlink_authentication_type_set	Sets the authentication type for the virtual interface
ospf_vlink_authentication_type_unset	Resets the authentication type for the virtual interface
ospf_vlink_dead_interval_set	Sets the router dead interval value for the virtual interface
ospf_vlink_dead_interval_unset	Resets the router dead interval value for the virtual interface
ospf_vlink_format_set	Specifies the format of the vlink address: decimal or A.B.C.D
ospf_vlink_hello_interval_set	Sets the router hello interval value for the virtual interface
ospf_vlink_hello_interval_unset	Resets the router hello interval value for the virtual interface
ospf_vlink_message_digest_key_set	Sets the MD5 authentication key for the virtual interface
ospf_vlink_message_digest_key_unset	Resets the MD5 authentication key for the virtual interface
ospf_vlink_retransmit_interval_set	Sets the retransmit interval value for the virtual interface
ospf_vlink_retransmit_interval_unset	Resets the retransmit interval value for the virtual interface (0)
ospf_vlink_set	Creates a virtual interface and configures a virtual neighbor

Function	Description
ospf_vlink_transmit_delay_set	Sets the transmit delay for the virtual link
ospf_vlink_transmit_delay_unset	Resets the transmit delay for the interface (0)
ospf_vlink_unset	Destroys the virtual interface and deconfigures the virtual neighbor

Include File

To call the functions in this section, include `ospfd/ospf_api.h`.

ospf_abr_type_set

This function sets the OSPF area border route (ABR) type.

This function is called by the following command:

```
ospf abr-type
```

Syntax

```
int
ospf_abr_type_set (u_int32_t vr_id, int proc_id, u_char type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	The type of area border router:
`cisco`	Cisco implementation
`ibm`	IBM implementation
`shortcut`	Shortcut ABR
`standard`	Standard behavior (RFC 2328)

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf_abr_type_unset

This function resets the configured ABR type; set the ABR type to CISCO implementation (default setting).

This function is called by the following command:

```
no ospf abr type
```

Syntax

```
int
ospf_abr_type_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_auth_type_set

This function enables authentication for an area.

This function is called by the following command:

```
area authentication
```

Syntax

```
int
ospf_area_auth_type_set (u_int32_t vr_id, int proc_id,
                         struct pal_in4_addr area_id, u_char type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`type`	The authentication type of area border router
`OSPF_AUTH_NULL`	No authentication required
`OSPF_AUTH_SIMPLE`	Simple password authentication
`OSPF_AUTH_CRYPTOGRAPHIC`	MD5 authentication

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AUTH_TYPE_INVALID when the authentication type is not supported

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

ospf_area_auth_type_unset

This function disables the authentication type for the area.

This function is called by the following command:

```
no area authentication
```

Syntax

```
int
ospf_area_auth_type_unset (u_int32_t vr_id, int proc_id,
                                  struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is out of range

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_default_cost_set

This function assigns the specified cost to the default summary route used for a not so stubby area (NSSA). This specifies the cost of sending a packet on an OSPF interface.

This function is called by the following command:

```
area default-cost
```


Syntax

```
int
ospf_area_default_cost_set (u_int32_t vr_id, int proc_id,
                            struct pal_in4_addr area_id, u_int32_t cost);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area identifier for the stub or NSSA: decimal value or IP address.
`cost`	The default cost for the area <0–16777215>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_COST_INVALID when the given cost is less than 0 or greater than 16777215

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_DEFAULT when the given area is a default area

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

ospf_area_default_cost_unset

This function resets the cost to the default value of 1.

This function is called by the following command:

```
no area default-cost
```

Syntax

```
int
ospf_area_default_cost_unset (u_int32_t vr_id, int proc_id,
                              struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID: decimal value or IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_DEFAULT when the given area is a default area

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

ospf_area_export_list_set

This function sets the type-3 export filter for networks announced to other areas.

This function is called by the following command:

```
area export list
```

Syntax

```
int
ospf_area_export_list_set (u_int32_t vr_id, int proc_id,
                           struct pal_in4_addr area_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`name`	The name of the access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

ospf_area_export_list_unset

This function resets the export list.

This function is called by the following command:

```
no area export
```

Syntax

```
int
```

```
ospf_area_export_list_unset (u_int32_t vr_id, int proc_id,
                             struct pal_in4_addr area_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_filter_list_access_set

This function sets to filter prefixes advertise in type-3 link-state advertisement (LSAs) with the access list name between OSPF areas of an ABR.

This function is called by the following command:

```
area filter-list access
```

Syntax

```
int
ospf_area_filter_list_access_set (u_int32_t vr_id, int proc_id,
                                  struct pal_in4_addr area_id,
                                        int type, char *name);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>
area_id	OSPF area ID
type	Filter type, one of the following
	FILTER_IN: filter list applied to prefixes advertised to the specified area from other areas
	FILTER_OUT: filter list applied to prefixes
name	Name of the access list

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_SUCCESS when the function succeeds

ospf_area_filter_list_access_unset

This function resets the filter list access configuration to either FILTER_IN or FILTER_OUT (selection is determined by type).

This function is called by the following command:

```
no area filter-list access
```

Syntax

```
int
ospf_area_filter_list_access_unset (u_int32_t vr_id, int proc_id,
                                    struct pal_in4_addr area_id, int type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`area_id`	OSPF area ID
`type`	Filter type, one of the following: FILTER_IN FILTER_OUT

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_area_filter_list_prefix_set

This function sets the type of filter prefix advertised in type-3 LSAs between the OSPF areas of an ABR.

This function is called by the following command:

```
area filter-list prefix
```

Syntax

```
int
ospf_area_filter_list_prefix_set (u_int32_t vr_id, int proc_id,
                                  struct pal_in4_addr area_id,
                                  int type, char *name)
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	OSPF area ID.
type	Filter type, one of the following: FILTER_IN FILTER_OUT
name	Name of the prefix list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_INVALID_FILTER_TYPE when the given filter type is not valid

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_SUCCESS when the function succeeds

ospf_area_filter_list_prefix_unset

This function resets the configuration: cancels the filter prefix advertise.

This function is called by the following command:

```
no area filter-list prefix
```

Syntax

```
int
ospf_area_filter_list_prefix_unset (u_int32_t vr_id, int proc_id,
                                    struct pal_in4_addr area_id, int type)
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	OSPF area ID.
type	Filter type, one of the following: FILTER_IN

FILTER_OUT

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_area_import_list_set

This function sets the import list value for the type-3 import filter.

This function is called by the following command:

```
area import-list
```

Syntax

```
int
ospf_area_import_list_set (u_int32_t vr_id, int proc_id,
                           struct pal_in4_addr area_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`name`	The name of the list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_import_list_unset

This function resets the import list value for the type-3 import.

This function is called by the following command:

```
no area import list
```

Syntax

```
int
ospf_area_import_list_unset (u_int32_t vr_id, int proc_id,
                             struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_no_summary_set

This function sets the OSPF area as stub. The stub area is extended to an all stubby area by configuring all of it as ABR.

This function is called by the following command:

```
area stub no summary
```

Syntax

```
int
ospf_area_no_summary_set (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_AREA_IS_DEFAULT when the given area is a default area

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_no_summary_unset

This function resets the `no-summary` restriction.

This function is called by the following command:

```
no area stub
```

Syntax

```
int
ospf_area_no_summary_unset (u_int32_t vr_id, int proc_id,
                            struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_DEFAULT when the given area is a default area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_default_originate_set

This function defines an area parameter as needed to be NSSA.

This function is called by the following command:

```
area <area ID> nssa
```

Syntax

```
int
ospf_area_nssa_default_originate_set (u_int32_t vr_id, int proc_id,
                                      struct pal_in4_addr area_id);
```


Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID: decimal value or IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_default_originate_unset

This function removes an area parameter as needed to be NSSA.

This function is called by the following command:

```
no area <area id> nssa
```

Syntax

```
int
ospf_area_nssa_default_originate_unset (u_int32_t vr_id, int proc_id,
                                        struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_no_redistribution_set

This function sets OSPF: redistribution in not allowed to the stub.

This function is called by the following command:

```
area <area id> nssa no-redistribution
```

Syntax

```
int
ospf_area_nssa_no_redistribution_set (u_int32_t vr_id, int proc_id,
                                      struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

ospf_area_nssa_no_redistribution_unset

This function resets OSPF: redistribution in not allowed to the stub.

This function is called by the following command:

```
no area no-redistribution
```

Syntax

```
int
ospf_area_nssa_no_redistribution_unset (u_int32_t vr_id, int proc_id,
                                        struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_set

This function defines an area as NSSA. This function is compiled if the HAVE_NSSA source code flag is turned on.

It is called by the following command:

```
area <area ID> nssa
```

Syntax

```
int
ospf_area_nssa_set (u_int32_t vr_id, int proc_id,
                    struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_HAS_VLINK when the area has a virtual link

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_area_nssa_stability_interval_set

This function sets the NSSA stability interval. If an elected translator determines its services are no longer required, it continues to perform its duties for this time interval. This minimizes excess flushing of translated Type-7 LSAs and provides a more stable translator transition.

It is called by the following commands:

```
area (A.B.C.D|<0-4294967295>) nssa stability-interval
no area (A.B.C.D|<0-4294967295>) nssa stability-interval
```

Syntax

```
int
ospf_area_nssa_stability_interval_set (u_int32_t vr_id, int proc_id,
                                       struct pal_in4_addr area_id, u_int32_t intvl)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`intvl`	NSSA stability interval in seconds <0-2147483647>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not an NSSA

OSPF_API_SET_SUCCESS when the function succeeds

ospf_area_nssa_translator_role_set

This function sets the ABR to be the translator between the types. This is applied only if the area type is NSSA.

This function is called by the following command:

```
area <area id> nssa translator-role
```

Syntax

```
int
ospf_area_nssa_translator_role_set (u_int32_t vr_id, int proc_id,
                                    struct pal_in4_addr area_id, u_char role);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

`area_id` OSPF area ID.

`role` NSSA-ABR translator role; one of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATE_CANDIDATE`
Translate NSSA-LSA to Type-5 LSA if router is elected

`OSPF_NSSA_TRANSLATE_ALWAYS`
Always translate NSSA-LSA to Type-5 LSA

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_translator_role_unset

This function removes the no-redistribution attribute. This function is applied only if the HAVE_NSSA flag is turned on.

It is called by the following command:

```
no area nssa translate-role
```

Syntax

```
int
ospf_area_nssa_translator_role_unset (u_int32_t vr_id, int proc_id,
                                      struct pal_in4_addr area_id);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`proc_id` The process ID for the OSPF instance <0–65535>.

`area_id` OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_nssa_unset

This function removes the NSSA designation from the specified area. This function is applied only if the HAVE_NSSA flag is turned on.

This function is called by the following command:

```
no area <area id> nssa
```

Syntax

```
int
ospf_area_nssa_unset (u_int32_t vr_id, int proc_id,
                      struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_range_not_advertise_set

This function sets the ABR to not advertise the summary LSA for each route in a specific adjacent area.

This function is called by the following command:

```
area <area_id> range <range_prefix> not-advertise
```

Syntax

```
int
ospf_area_range_not_advertise_set (u_int32_t vr_id, int proc_id,
                                   struct pal_in4_addr area_id,
                                   struct pal_in4_addr net, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_AREA_RANGE_NOT_EXIST when the specified area range does not exist

ospf_area_range_not_advertise_unset

This function allows the area border router (ABR) to create a summary LSA for each route in specific area and advertise it in adjacent areas.

This function is called by the following command:

```
no area range not-advertise
```

Syntax

```
int
ospf_area_range_not_advertise_unset (u_int32_t vr_id, int proc_id,
                                     struct pal_in4_addr area_id,
                                           struct pal_in4_addr net, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length from the `/M` sub-parameter of the `net` parameter.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_AREA_RANGE_NOT_EXIST when the specified area range does not exist

ospf_area_range_set

This function specifies an address range, for which to advertise a single route to other areas by the ABRs.

This function is called by the following command:

```
area <area_id> range <net/masklen>
```

Syntax

```
int
ospf_area_range_set(u_int32_t vr_id, int proc_id, struct pal_in4_addr area_id,
                    struct pal_in4_addr net, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length from the `/M` sub-parameter of the `net` parameter.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

ospf_area_range_substitute_set

This function summarizes routes via the following:

- Matching addresses and or masks, border routers only
- Announcing area range as a prefix

This function is called by the following command:

```
area <area_id> range <net/masklen> substitute <subst_net/subst_masklen>
```

Syntax

```
int
ospf_area_range_substitute_set (u_int32_t vr_id, int proc_id,
                                struct pal_in4_addr area_id,
                                struct pal_in4_addr net, u_char masklen,
                                struct pal_in4_addr subst_net,
                                u_char subst_masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length from the `/M` sub-parameter of the `net` parameter.
`subst_net`	The substitute network.
`subst_masklen`	The substitute prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_range_substitute_unset

This function resets the configuration.

This function is called by the following command:

```
no area range substitute
```

Syntax

```
int
ospf_area_range_substitute_unset (u_int32_t vr_id, int proc_id,
                                  struct pal_in4_addr area_id,
                                  struct pal_in4_addr net, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length from the `/M` sub-parameter of the `net` parameter.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_AREA_RANGE_NOT_EXIST when the given area range cannot be found

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

ospf_area_range_unset

This function deletes the area range.

This function is called by the following command:

```
no area range
```

Syntax

```
int ospf_area_range_unset (u_int32_t vr_id, int proc_id,
                           struct pal_in4_addr area_id,
                           struct pal_in4_addr net, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`net`	The network address range.
`masklen`	The mask length from the `/M` sub-parameter of the `net` parameter.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_RANGE_NOT_EXIST when the given range does not exist

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_area_shortcut_set

This function sets the shortcut mode of the specified area. Area shortcut enables traffic to go through the non-backbone area with a lower metric, whether or not the ABR is attached to the backbone area.

This function is called by the following command:

```
area <area_id> shortcut (default|enable|disable)
```

Syntax

```
int
ospf_area_shortcut_set (u_int32_t vr_id, int proc_id,
                        struct pal_in4_addr area_id, u_char type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.
`type`	The type of shortcut:
`default`	Sets default short cutting behavior.
`enable`	Forces short-cutting through the area.
`disable`	Disables short-cutting through the area.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is on the backbone

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

ospf_area_shortcut_unset

This function removes the shortcut mode of the specified area.

This function is called by the following command:

```
no area shortcut
```

Syntax

```
int
ospf_area_shortcut_unset (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is on the backbone

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_area_stub_set

This function sets the specified area as a stub area.

This function is called by the following command:

```
area <area_id> stub
```

Syntax

```
int
ospf_area_stub_set (u_int32_t vr_id, int proc_id,
                    struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is on the backbone

OSPF_API_SET_ERR_AREA_HAS_VLINK when the area has a virtual link

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_area_stub_unset

This function removes the stub definition from the specified area.

This function is called by the following command:

```
no area <area_id> stub
```

Syntax

```
int
ospf_area_stub_unset (u_int32_t vr_id, int proc_id,
                      struct pal_in4_addr area_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	OSPF area ID.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is on the backbone

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_auto_cost_reference_bandwidth_set

This function sets the reference bandwidth value. OSPF calculates the OSPF metric for an interface by dividing the reference bandwidth.

This function is called by the following command:

```
auto-cost reference-bandwidth <reference bandwith>
```

Syntax

```
int
ospf_auto_cost_reference_bandwidth_set (u_int32_t vr_id,
                                        int proc_id, int refbw);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
refbw	The bandwidth in Mbits/second <1–4294967>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_REFERENCE_BANDWIDTH_INVALID when the bandwidth is not within the range

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_auto_cost_reference_bandwidth_unset

This function disables the bandwidth restriction.

This function is called by the following command:

```
no auto-cost
```

Syntax

```
int
ospf_auto_cost_reference_bandwidth_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_bfd_all_interfaces_set

This function enables Bidirectional Forwarding Detection (BFD) on all interfaces.

This function is called by the following command:

```
bfd all-interfaces
```

Syntax

```
int
ospf_bfd_all_interfaces_set (u_int32_t vr_id, int proc_id);
```


Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_bfd_all_interfaces_unset

This function disables Bidirectional Forwarding Detection (BFD) on all interfaces.

This function is called by the following command:

```
no bfd all-interfaces
```

Syntax

```
int
ospf_bfd_all_interfaces_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_capability_cspf_set

This function sets the Constrained Shortest Path First (CSPF) capability for an OSPF process. The CSPF protocol module relies on the OSPF database to calculate the shortest path through the network.

This function is called by the following command:

```
capability cspf
```

Syntax

```
int
ospf_capability_cspf_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_ERR_CSPF_INSTANCE_EXIST when the specified CSPF instance does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_capability_cspf_unset

This function disables the CSPF capability for an OSPF process.

This function is called by the following command:

```
no capability
```

Syntax

```
int
ospf_capability_cspf_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_ERR_CSPF_INSTANCE_EXIST when the specified CSPF instance does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_capability_opaque_lsa_set

This function sets the opaque capability for an OSPF process.

This function is called by the following command:

```
capability opaque
```

Syntax

```
int
ospf_capability_opaque_lsa_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_capability_opaque_lsa_unset

This function unsets the opaque capability for an OSPF process.

This function is called by the following command:

```
no capability opaque
```

Syntax

```
int
ospf_capability_opaque_lsa_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_capability_traffic_engineering_set

This function sets the traffic engineering capability for an OSPF process.

This function is called by the following command:

```
capability te
```

Syntax

```
int
ospf_capability_traffic_engineering_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_capability_traffic_engineering_unset

This function resets the traffic engineering capability for an OSPF process.

This function is called by the following command:

```
no capability te
```

Syntax

```
int
ospf_capability_traffic_engineering_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router cannot be found

ospf_compatible_rfc1583_set

This function calculates route summary costs according to RFC 1583.

This function is called by the following command:

```
compatible rfc1583
```

Syntax

```
int
ospf_compatible_rfc1583_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_compatible_rfc1583_unset

This function disables the calculation of route summary costs according to RFC 1583.

This function is called by the following command:

```
no compatible rfc1583
```

Syntax

```
int
ospf_compatible_rfc1583_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_default_metric_set

This function sets the default metric for redistributed routes.

This function is called by the following command:

```
default-metric <metric>
```

Syntax

```
int
ospf_default_metric_set (u_int32_t vr_id, int proc_id, int metric);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`metric`	The default metric of redistributed routes: 1-16777214.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_METRIC_INVALID when the given metric value is not within the range

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_default_metric_unset

This function disables the default metric for the specified process.

This function is called by the following command:

```
no default metric
```

Syntax

```
int
ospf_default_metric_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_disable_db_summary_opt

This function disables the OSPF Database Summary List optimization. The `OSPF_DB_SUMMARY_OPT` flag is unset to indicate the feature is disabled.

This function is called by the following command:

```
no enable db-summary-opt
```

Syntax

```
int
ospf_disable_db_summary_opt (u_int32_t vr_id,
int area border router (ABR)proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Values

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF_API_SET_SUCCESS when the action succeeds

ospf_disable_ext_multi_inst

This function disables the support of multiple OSPF instances on a subnet.

This function is called by the following command:

```
no enable ext-ospf-multi-inst
```

Syntax

```
s_int8_t
ospf_disable_ext_multi_inst (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_EXT_MULTI_INST_NOT_ENABLED if multiple-instance support is not enabled

ospf_distance_all_set

This function sets the specified value of administrative distance for all route types: external, inter-area and intra- area.

This function is called by the following command:

```
distance <administrative_distance>
```

Syntax

```
int
ospf_distance_all_set (u_int32_t vr_id, int proc_id, int distance);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`distance`	The administrative distance value of the route <1–255>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_DISTANCE_INVALID when the distance value is outside the range

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_all_unset

This function resets all route administrative distances: intra-area, inter-area and external.

This function is called by the following command:

```
no distance <OSPF Administrative distance>
```

Syntax

```
int
```

```
ospf_distance_all_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_external_set

This function sets the specified administrative distance of the external route.

This function is called by the following command:

```
distance ospf external <distance>
```

Syntax

```
int
ospf_distance_external_set (u_int32_t vr_id, int proc_id, int distance);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`distance`	The administrative distance value of the route <1–255>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_DISTANCE_INVALID when the distance value is outside the range

ospf_distance_external_unset

This function resets the route administrative distance value to 0.

This function is called by the following command:

```
no distance ospf
```

Syntax

```
int
ospf_distance_external_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_inter_area_set

This function sets the specified distance value for inter-area routes.

This function is called by the following command:

```
distance ospf inter-area <distance>
```

Syntax

```
int
ospf_distance_inter_area_set (u_int32_t vr_id, int proc_id, int distance);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`distance`	The administrative distance value of the route <1–255>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_DISTANCE_INVALID when the distance value is outside the range

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_inter_area_unset

This function resets the route administrative distance value to 0.

This function is called by the following command:

```
no distance ospf
```

Syntax

```
int
ospf_distance_inter_area_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_intra_area_set

This function sets the specified distance value of the route.

This function is called by the following command:

```
distance ospf intra-area <distance>
```

Syntax

```
int
ospf_distance_intra_area_set (u_int32_t vr_id, int proc_id, int distance);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`distance`	The administrative distance value of the route <1–255>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_DISTANCE_INVALID when the distance value is outside the range

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_intra_area_unset

This function resets the route administrative distance value to 0.

This function is called by the following command:

```
no distance ospf
```

Syntax

```
int
ospf_distance_intra_area_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distance_source_set

This function sets the administrative distance to prefixes whose nexthop matches the given source IP address.

This function is called by the following command:

```
distance <1-255> A.B.C.D/M (WORD|)
```

Syntax

```
int
ospf_distance_source_set (u_int32_t vr_id, int proc_id,
                          u_int32_t distance, struct pal_in4_addr addr,
                          u_char masklen, char *access_name)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`proc_id`	The process ID for the OSPF instance <0–65535>.
`distance`	The administrative distance value of the route <1–255>.
`addr`	IP source prefix.

masklen	Length of mask.
access_name	Name of access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_MALLOC_ERR when there is an error allocating memory

OSPF_API_SET_SUCCESS when the function succeeds

ospf_distance_source_unset

This function unsets the administrative distance to prefixes whose nexthop matches the given source IP address.

This function is called by the following command:

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

Syntax

```
int
ospf_distance_source_unset (u_int32_t vr_id, int proc_id,
                            struct pal_in4_addr addr,
                            u_char masklen, char *access_name)
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
addr	IP source prefix.
masklen	Length of mask.
access_name	Name of access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_DISTANCE_NOT_EXIST when the configured distance is NULL

OSPF_API_SET_SUCCESS when the function succeeds

ospf_distribute_list_in_set

This function sets the inbound distribute list for the specified protocol.

This function is called by the following command:

```
distribute-list <access-list name> in
```

Syntax

```
int
ospf_distribute_list_in_set (u_int32_t vr_id, int proc_id,
                             char *name)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	The type of route to filter; one of the constants from Protocol Constants
`name`	The name of the access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_MALLOC_ERR when memory allocation fails

OSPF_API_SET_SUCCESS when the function succeeds

ospf_distribute_list_out_unset

This function disables the inbound distribute list.

This function is called by the following command:

```
no distribute list <access-list name> in
```

Syntax

```
int
ospf_distribute_list_in_unset (u_int32_t vr_id, int proc_id,
                               char *name)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`name`	The name of the access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_distribute_list_out_set

This function sets the outbound distribute list for the specified protocol.

This function is called by the following command:

```
distribute-list <access-list name> out
```

Syntax

```
int
ospf_distribute_list_out_set (u_int32_t vr_id, int proc_id,
                              int type, int sproc_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`sproc_id`	The process ID for the OSPF instance used for redistribution from one OSPF instance to another <1–65535>.
`type`	The type of route to filter; one of the constants from Protocol Constants
`name`	The name of the access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the protocol is not valid

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_distribute_list_out_unset

This function disables the outbound distribute list for the specified protocol.

This function is called by the following command:

```
no distribute list <access-list name> out
```

Syntax

```
int
ospf_distribute_list_out_unset (u_int32_t vr_id, int proc_id,
                                int type, int sproc_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	The type of route to be filtered; one of the constants from Protocol Constants
`sproc_id`	The process ID for the OSPF instance used for redistribution from one OSPF instance to another <1–65535>.
`name`	The name of the access list.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the protocol is not supported

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_domain_id_set

This function sets an OSPF domain ID as specified: domain ID for a particular OSPF VRF instance.

This function is called by the following command:

`domain-id` <OSPF domain type> value <ospf domain ID>

Syntax

```
int
ospf_domain_id_set (u_int32_t vr_id, int proc_id, char *type,
                    u_int8_t *value, bool_t is_primary_did);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	Domain ID type, one of the following:
`type-as`	AS format
`type-as4`	AS4 format
`type-back-comp`	Backward compatibility

Note: Type can also be NULL in the case of a NULL domain ID or IP address format domain ID.

`value` Domain ID value entered through the CLI.

`is_primary_did` Boolean flag to identify whether or not the entered domain ID is primary.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_SEC_DOMAIN_ID when the secondary ID is configured before the primary ID

OSPF_API_SET_ERR_NON_ZERO_DOMAIN_ID when no null domain ID exists

OSPF_API_SET_ERR_WRONG_VALUE when an invalid value is given

OSPF_API_SET_ERR_INVALID_HEX_VALUE when an invalid hex value is given

OSPF_API_SET_ERR_DUPLICATE_DOMAIN_ID when a duplicate ID is configured

ospf_domain_id_unset

This function removes the OSPF domain ID.

This function is called by the following command:

```
no domain-id type <ospf domain ID type> value <Domain ID value>
```

Syntax

```
int
ospf_domain_id_unset (u_int32_t vr_id, int proc_id, char *type,
                      u_int8_t *value, bool_t is_primary_did);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`proc_id` The process ID for the OSPF instance <0–65535>.

`type` Domain ID type, one of the following:

- `type-as` AS format
- `type-as4` AS4 format
- `type-back-comp` Backward compatibility

Note: Type can also be NULL in the case of a NULL domain ID or IP address format domain ID.

`value` Domain ID value entered through the CLI.

`is_primary_did` Boolean flag to identify whether or not the entered domain ID is primary.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_WRONG_VALUE when an invalid value is given

OSPF_API_SET_ERR_INVALID_HEX_VALUE when an invalid hex value is given

ospf_dna_set

This function enables flood reduction feature on all OSPF enabled interfaces

This function is called by the following command:

```
ospf flood-reduction
```

Syntax

```
int
ospf_dna_set (struct ipi_vr *vr, int proc_id);
```

Input Parameters

`vr`	Pointer to the Virtual router Parameters
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

ospf_dna_unset

This function disables flood reduction feature on all OSPF enabled interfaces

This function is called by the following command:

```
no ospf flood-reduction
```

Syntax

```
int
ospf_dna_unset (struct ipi_vr *vr, int proc_id);
```

Input Parameters

`vr`	Virtual router Parameters

`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

ospf_enable_db_summary_opt

This function enables the OSPF Database Summary List optimization. The `OSPF_DB_SUMMARY_OPT` flag is set to indicate the feature is enabled.

This function is called by the following command:

```
enable db-summary-opt
```

Syntax

```
int
ospf_enable_db_summary_opt (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Values

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is invalid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF_API_SET_SUCCESS when the action succeeds

ospf_enable_ext_multi_inst

This function enables multiple OSPF instances to run on a subnet.

This function is called by the following command:

```
enable ext-ospf-multi-inst
```

Syntax

```
s_int8_t
ospf_enable_ext_multi_inst (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_key_set

This function sets the authentication key for “simple password”.

The configuration is stored regardless of whether a real interface exists.

This function is called by the following command:

```
ip ospf authentication-key
```

Syntax

```
int
ospf_if_authentication_key_set (u_int32_t vr_id, char *name, char *auth_key);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`auth_key`	The authentication key, null-terminated.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_key_set_by_addr

This function sets the interface's authentication key for an area of the specified IP address with a simple password.

The configuration is stored regardless of whether a real interface exists.

This function is called by the following command:

```
ip ospf authentication-key
```

Syntax

```
int
ospf_if_authentication_key_set_by_addr (u_int32_t vr_id, char *name,
                                        struct pal_in4_addr addr,
                                        char *auth_key)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

`name`	The interface name.
`addr`	The IP address of the interface.
`authkey`	The authentication key, null terminated.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_key_unset

This function removes the interface authentication key for an area.

This function is called by the following command:

```
no ip ospf authentication-key
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when OSPF interface parameters are not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_key_unset_by_addr

This function disables the authentication key for the interface specified by IP address.

This function is called by the following command:

```
no ip ospf authentication-key
```

Syntax

```
int
ospf_if_authentication_key_unset_by_addr (u_int32_t vr_id, char *name,
                                          struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the key is not set

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_type_set

This function sets the authentication type of the current interface. The configuration is stored regardless of whether a real interface exists or not.

This function is called by the following command:

```
ip ospf authentication
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`type`	Authentication type:
`(0)`	None
`(1)`	Simple Password
`(2)`	Cryptographic

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AUTH_TYPE_INVALID when the given authentication type is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_type_set_by_addr

This function sets the authentication type of the interface specified by IP address.

The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
ip ospf authentication
```

Syntax

```
int
ospf_if_authentication_type_set_by_addr (u_int32_t vr_id, char *name,
                                         struct pal_in4_addr addr,
                                         u_char type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`type`	Authentication type, select one of the following digits:
`0`	None
`1`	Simple Password
`2`	Cryptographic

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_AUTH_TYPE_INVALID when the given authentication type is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_type_unset

This function removes the authentication type for the current interface.

This function is called by the following command:

```
no ip ospf authentication
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_authentication_type_unset_by_addr

This function resets the authentication type for the specified interface.

This function is called by the following command:

```
no ip ospf authentication
```

Syntax

```
int
ospf_if_authentication_type_unset_by_addr (u_int32_t vr_id, char *name,
                                           struct pal_in4_addr addr);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
name	The interface name.
addr	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_cost_set

This function sets the current interface output cost. The configuration is stored regardless of whether or not a real interface exists.

This function is called by the following command:

```
ip ospf cost
```

Syntax

```
int
ospf_if_cost_set (u_int32_t vr_id, char *name, u_int32_t cost);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
name	The interface name.
cost	The output cost for the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_COST_INVALID when the given cost is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_cost_set_by_addr

This function sets the output cost of the interface of the specific IP address. The configuration is stored regardless of whether or not a real interface exists.

This function is called by the following command:

```
ip ospf cost
```

Syntax

```
int
ospf_if_cost_set_by_addr (u_int32_t vr_id, char *name,
                          struct pal_in4_addr addr, u_int32_t cost);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`cost`	The output cost for the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_COST_INVALID when the given cost is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_cost_unset

This function resets the cost for the current interface to the default value: 10.

This function is called by the following command:

```
no ip ospf cost
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_cost_unset_by_addr

This function resets the cost for the specified interface to the default value: 10.

This function is called by the following command:

```
no if ospf cost
```

Syntax

```
int
ospf_if_cost_unset_by_addr (u_int32_t vr_id, char *name,
                            struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

ospf_if_database_filter_set

This function suppresses all LSA during synchronization and flooding on a particular interface.

This function is called by the following command:

```
ip ospf database-filter
```

Syntax

```
int
```

```
ospf_if_database_filter_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_database_filter_set_by_addr

This function suppresses all LSA during synchronization and flooding for a particular interface by IP address.

This function is called by the following command:

```
ip ospf database-filter
```

Syntax

```
int
ospf_if_database_filter_set_by_addr (u_int32_t vr_id, char *name,
                                     struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	IP interface address.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_database_filter_unset

This function restores the forwarding of LSAs.

This function is called by the following command:

```
no ip ospf database-filter
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

ospf_if_database_filter_unset_by_addr

This function restores the database filter of the interface specified by IP address.

This function is called by the following command:

```
no ip ospf database-filter
```

Syntax

```
int
ospf_if_database_filter_unset_by_addr (u_int32_t vr_id, char *name,
                                       struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	IP interface address.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface parameters are not configured

ospf_if_dc_set

This function enables demand circuit feature on the interface by setting the DC_ENABLE flag.

This function is called by the following command:

```
ip ospf demand-circuit
```

Syntax

```
int
```

```
ospf_if_dc_set (u_int32_t vr_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_MALLOC_ERR when the specified interface is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_if_dc_unset

This function disables demand-circuit feature on an interfaces.

This function is called by the following command:

```
no ip ospf demand-circuit
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_MALLOC_ERR when the specified interface is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_if_dead_interval_set

This function sets the router-dead-interval for the current interface. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
ip ospf dead-interval
```

Syntax

```
int
```

```
ospf_if_dead_interval_set (u_int32_t vr_id, char *name, u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_DEAD_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_dead_interval_set_by_addr

This function sets the router-dead-interval for the interface specified by the IP address. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
ip ospf dead-interval
```

Syntax

```
int
ospf_if_dead_interval_set_by_addr (u_int32_t vr_id, char *name,
                                   struct pal_in4_addr addr,
                                   u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_DEAD_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_dead_interval_unset

This function resets the dead interval of the specified interface to the default specified by the `OSPF_IF_PARAM_DEAD_INTERVAL_DEFAULT` macro in `ospfd/ospf_interface.h`.

This function is called by the following command:

```
no ip ospf dead-interval
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_dead_interval_unset_by_addr

This function resets the dead interval of the interface specified by IP address to the default specified by `OSPF_IF_PARAM_DEAD_INTERVAL_DEFAULT` macro in `ospfd/ospf_interface.h`.

This function is called by the following command:

```
no ip ospf dead-interval
```

Syntax

```
int
ospf_if_dead_interval_unset_by_addr (u_int32_t vr_id, char *name,
                                     struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_disable_all_set

This function disables all packet processing on a particular interface regardless whether the network area command is configured.

This function is called by the following command:

```
ip ospf disable all
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_disable_all_unset

This function unconfigures the `ip ospf disable all` command.

This function is called by the following command:

```
no ip ospf disable all
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_dna_set

This function enables flood reduction feature on an interface.

This function is called by the following command:

```
ip ospf flood-reduction
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_MALLOC_ERR when the specified interface is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_if_dna_unset

This function disables flood reduction feature on an interface.

This function is called by the following command:

```
no ip ospf flood-reduction
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_MALLOC_ERR when the specified interface is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_if_hello_interval_set

This function sets the hello interval for the current interface. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
ip ospf hello-interval
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_hello_interval_set_by_addr

This function sets the hello interval for the interface specified by IP address. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
ip ospf hello-interval
```

Syntax

```
int
ospf_if_hello_interval_set_by_addr (u_int32_t vr_id, char *name,
                                    struct pal_in4_addr addr,
                                    u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_hello_interval_unset

This function resets the hello interval of the current interface to the default value specified by OSPF_HELLO_INTERVAL_DEFAULT (refer to ospf_interface.h).

This function is called by the following command:

```
no ip ospf hello-interval
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_hello_interval_unset_by_addr

This function resets the hello interval of the interface specified by IP address to the default value `OSPF_HELLO_INTERVAL_DEFAULT` (refer to ospf_interface.h).

This function is called by the following command:

```
no ip ospf hello-interval
```

Syntax

```
int
ospf_if_hello_interval_unset_by_addr (u_int32_t vr_id, char *name,
                                      struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_conf_ldp_igp_sync

This function enables LDP-OSPF synchronization on the current interface.

Syntax

```
int
ospf_if_conf_ldp_igp_sync (u_int32_t vr_id, struct interface *ifp,
                           u_int32_t holddown_timer)
```

Input Parameters

`vr_id`	Virtual router ID
`ifp`	Pointer to the interface on which LDP-OSPF synchronization is enabled
`holddown timer`	Sets the hold-down timer for synchronization

Output Parameters

None

Return Values

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_VR_NOT_EXIST when the OSPF master is not present

OSPF_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

OSPF_API_SET_ERR_IF_SYNC_NOT_EXIST when call failed to retrieve existing synchronization parameters or to allocate memory for new parameters

OSPF_API_SET_ERR_IF_SYNC_EXIST when synchronization is re-configured with same hold-down timer

OSPF_API_SET_ERROR when LDP session state query was not successfully sent

ospf_if_conf_ldp_igp_unsync

This call disables LDP-OSPF synchronization on the current interface.

Syntax

```
int
ospf_if_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-OSPF synchronization is disabled

Output Parameters

None

Return Values

OSPF_API_SET_SUCCESS when the call is successful

OSPF_API_SET_ERR_VR_NOT_EXIST when the OSPF master is not present

OSPF_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

ospf_if_message_digest_key_set

This function sets the MD5 key for the current interface. The configuration is stored regardless of whether the interface exists.

This function is called by the following command:

```
ip ospf message-digest-key
```

Syntax

```
int
ospf_if_message_digest_key_set (u_int32_t vr_id, char *name, u_char key_id,
                                char *auth_key);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`key_id`	The key ID.
`auth_key`	The MD 5 key. This is a null terminated value.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_MD5_KEY_EXIST when the MD5 key already exists

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_message_digest_key_set_by_addr

This function sets the MD5 key for the interface specified by IP address. The configuration is stored regardless of whether the interface exists.

This function is called by the following command:

```
ip ospf message-digest-key
```

Syntax

```
int
```

```
ospf_if_message_digest_key_set_by_addr (u_int32_t vr_id, char *name,
                                        struct pal_in4_addr addr,
                                        u_char key_id, char *auth_key);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`key_id`	The key ID.
`auth_key`	The MD 5 key. This is a null terminated value.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_MD5_KEY_EXIST when the MD5 key already exists

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_message_digest_key_unset

This function deletes the MD5 key for the current interface. The configuration is stored regardless of whether the interface exists.

This function is called by the following command:

```
no ip ospf message-digest-key
```

Syntax

```
int
ospf_if_message_digest_key_unset (u_int32_t vr_id, char *name, u_char key_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`key_id`	The key ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_MD5_KEY_NOT_EXIST when the MD5 key does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_message_digest_key_unset_by_addr

This function deletes the MD5 key for the interface specified by IP address. The configuration is stored regardless of whether the interface exists.

This function is called by the following command:

```
no ip ospf message-digest-key
```

Syntax

```
int
ospf_if_message_digest_key_unset_by_addr (u_int32_t vr_id, char *name,
                                          struct pal_in4_addr addr,
                                          u_char key_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`key_id`	The key ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_MD5_KEY_NOT_EXIST when the MD5 key does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_network_p2mp_nbma_set

This function configures an interface to Point-to-Multipoint Non-Broadcast mode.

This function is called by the following command:

```
ip ospf network point-to-multipoint non-broadcast
```

Syntax

```
int
ospf_if_network_p2mp_nbma_set (u_int32_t vr_id, char *name);
```

Input parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_network_set

This function configures the OSPF network type as specified.

This function is called by the following command:

```
ip ospf network-type
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`type`	The network type:
	Point-to-Point
	Broadcast
	NBMA
	Point-to-MultiPoint

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_NETWORK_TYPE_INVALID when the network type is not supported

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_network_unset

This function resets the network type to the default type (OSPF_IFTYPE_BROADCAST) (refer to ospf_interface.h).

This function is called by the following command:

```
no ip ospf network-type
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_priority_set

This function sets the priority of the current interface.

This function is called by the following command:

```
ip ospf priority <priority>
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`priority`	The router priority: <0–255>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_priority_set_by_addr

This sets the priority of the interface specified by IP address.

This function is called by the following command:

```
ip ospf priority
```

Syntax

```
int
ospf_if_priority_set_by_addr (u_int32_t vr_id, char *name,
                              struct pal_in4_addr addr, u_char priority);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

`addr`	The IP address of the interface.
`priority`	The router priority of the interface: <0–255>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_priority_unset

This function resets the priority of the current interface to the default value: 1.

This function is called by the following command:

```
no ip ospf priority
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_priority_unset_by_addr

This function resets the priority of the interface specified by IP address of the default value: 1.

This function is called by the following command:

```
no ip ospf priority <address>
```

Syntax

```
int
ospf_if_priority_unset_by_addr (u_int32_t vr_id, char *name,
                                struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_resync_timeout_set

This function sets the timeout interval for re-synchronization. If out-of-band re-synchronization does not occur, adjacency is reset.

This function is called by the following command:

```
ip ospf (A.B.C.D|) resync-timeout <1-65535>
```

Syntax

```
int
ospf_if_resync_timeout_set (u_int32_t vr_id, char *name, u_int32_t timeout);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`timeout`	The re-synchronization timeout in seconds: <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_resync_timeout_unset

This function resets the priority of the interface specified by IP address of the default value: 1.

This function is called by the following command:

```
no ip ospf priority <address>
```

Syntax

```
int
ospf_if_priority_unset_by_addr (u_int32_t vr_id, char *name,
                                struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_retransmit_interval_set

This function sets the time between LSA retransmission for adjacencies belonging to the interface.

After sending an LSA to a neighbor, the router keeps the LSA until it receives an acknowledgement. If the router does not receive an acknowledgement during the set time (the retransmit interval value), it retransmits the LSA.

This function is called by the following command:

```
ip ospf retransmit-interval
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_retransmit_interval_set_by_addr

This function the time between LSA retransmission for adjacencies belonging to the interface by ip address.

This function is called by the following command:

```
ip ospf retransmit-interval
```

Syntax

```
int
ospf_if_retransmit_interval_set_by_addr (u_int32_t vr_id, char *name,
                                         struct pal_in4_addr addr,
                                         u_int32_t interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`interval`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_retransmit_interval_unset

This function resets the retransmit interval of the current interface to the default specified by OSPF_RETRANSMIT_INTERVAL_DEFAULT.

This function is called by the following command:

```
no ip ospf retransmit-interval
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_retransmit_interval_unset_by_addr

This function resets the retransmit interval of the interface specified by IP address to the default specified by `OSPF_RETRANSMIT_INTERVAL_DEFAULT` (refer to ospf_interface.h).

This function is called by the following command:

```
no ip ospf retransmit-interval
```

Syntax

```
int
ospf_if_retransmit_interval_unset_by_addr (u_int32_t vr_id, char *name,
                                           struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_te_metric_set

This function sets the TE-metric on the specified interface.

This function is called by the following command:

```
te-metric
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`metric`	TE metric <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_IF_COST_INVALID when the given metric is outside of the range

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_te_metric_unset

This function unsets the TE-metric on a particular interface.

This function is called by the following command:

`no te-metric` command

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_transmit_delay_set

This function sets the transmit delay interval (seconds) for the current interface. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

`ip ospf transmit-delay`

Syntax

```
int
ospf_if_transmit_delay_set (u_int32_t vr_id, char *name, u_int32_t delay);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`delay`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_transmit_delay_set_by_addr

This function sets the transmit delay interval (seconds) for the interface specified by IP address. The configuration is stored regardless of whether the real interface exists.

This function is called by the following command:

```
no ip ospf transmit-delay
```

Syntax

```
int
ospf_if_transmit_delay_set_by_addr (u_int32_t vr_id, char *name,
                                    struct pal_in4_addr addr,
                                    u_int32_t delay);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.
`delay`	The interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when the given interval is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_transmit_delay_unset

This function resets the transmit delay interval of the current interface to the default specified by OSPF_TRANSMIT_DELAY_DEFAULT (refer to ospf_interface.h).

This function is called by the following command:

```
no ip ospf transmit-delay
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_if_transmit_delay_unset_by_addr

This function resets the transmit delay interval of the interface specified by IP address to the default specified by OSPF_TRANSMIT_DELAY_DEFAULT (refer to `ospf_interface.h`).

This function is called by the following command:

```
no ip ospf transmit-delay
```

Syntax

```
int
ospf_if_transmit_delay_unset_by_addr (u_int32_t vr_id, char *name,
                                      struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_lsa_min_arrival_set

This function sets the minimum interval to accept the same link-state advertisement (LSA) from OSPF neighbors.

This function is called by the following command:

```
timers lsa arrival
```

Syntax

```
int
ospf_lsa_min_arrival_set (u_int32_t vr_id, int proc_id,
                          u_int32_t min_arrival)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`min_arrival`	The minimum delay in milliseconds between accepting the same LSA from neighbors <0-600000>

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_lsa_min_arrival_unset

This function sets the minimum interval to accept the same link-state advertisement (LSA) from OSPF neighbors to its default value (1000 milliseconds).

This function is called by the following command:

```
no timers lsa arrival
```

Syntax

```
int
ospf_lsa_min_arrival_unset (u_int32_t vr_id, int proc_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_lsa_throttle_timers_set

This function sets the rate-limiting intervals for OSPF link-state advertisement (LSA) generation.

This function is called by the following command:


```
timers throttle
```

Syntax

```
int
ospf_lsa_throttle_timers_set (u_int32_t vr_id, int proc_id,
                              u_int32_t start_delay, u_int32_t hold_interval,
                              u_int32_t max_delay)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`start_delay`	The minimum delay in milliseconds for the generation of LSAs <0-600000>. The first instance of LSA is always generated immediately upon a local OSPF topology change. The generation of the next LSA is not before the start interval.
`hold_interval`	The hold time in milliseconds <0-600000>. This value is used to calculate the subsequent rate limiting times for LSA generation.
`max_delay`	The maximum wait time in milliseconds between generation of the same LSA <0-600000>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_TIMER_VALUE_INVALID when one of the given intervals is invalid

OSPF_API_SET_SUCCESS when the function succeeds

ospf_lsa_throttle_timers_unset

This function sets the rate-limiting intervals for OSPF link-state advertisement (LSA) generation to their default values:

- Default start interval: 0 milliseconds
- Default hold interval: 5000 milliseconds
- Default maximum interval: 5000 milliseconds

This function is called by the following command:

```
no timers throttle
```

Syntax

```
int
ospf_lsa_throttle_timers_unset (u_int32_t vr_id, int proc_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_max_area_limit_set

This function sets the maximum number of OSPF areas.

This function is called by the following command:

```
maximum-area
```

Syntax

```
int
ospf_max_area_limit_set (struct ospf *top, u_int32_t num);
```

Input Parameters

`top`	Pointer to the OSPF master structure
`num`	Maximum number of OSPF areas <1–4294967294>

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

ospf_max_area_limit_unset

This function resets the maximum number of OSPF areas.

This function is called by the following command:

```
no maximum-area
```

Syntax

```
int
ospf_max_area_limit_unset (struct ospf *top)
```

Input Parameters

`top`	Pointer to the OSPF master structure

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

ospf_max_concurrent_dd_set

This function sets the specified limit for the number of concurrently processed Database Descriptors (DD).

This function is called by the following command:

```
max-concurrent-dd
```

Syntax

```
int
ospf_max_concurrent_dd_set (u_int32_t vr_id, int proc_id, u_int16_t max_dd)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`max_dd`	Maximum Database Descriptor (DD) processes <1–65535>

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_max_concurrent_dd_unset

This function resets the specified limit for the number of concurrently processed Database Descriptors (DD).

This function is called by the following command:

```
no max-concurrent-dd
```

Syntax

```
int
ospf_max_concurrent_dd_unset (u_int32_t vr_id, int proc_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

ospf_multi_area_adjacency_unset

This function disables multi-area-adjacency.

This function is called by the following command:

```
no area multi-area-adjacency
```

Syntax

```
s_int32_t
ospf_multi_area_adjacency_unset (u_int32_t vr_id, int proc_id,
                                 struct pal_in4_addr area_id, u_char *ifname,
                                 struct pal_in4_addr nbr_addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The OSPF area ID.
`ifname`	The interface name.
`nbr_addr`	The neighbor IP address: A.B.C.D.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_MULTI_AREA_ADJ_NOT_SET when the multi-area adjacency is not assigned for the particular area on the interface

ospf_nbr_static_config_check

This function checks the neighbor configuration state.

Note: For the multi-area adjacency feature, this function checks for the neighbor configuration state for multi-area networks.

Syntax

```
int
ospf_nbr_static_config_check (u_int32_t vr_id, int proc_id,
```



```
                                struct pal_in4_addr addr, int config);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address: A.B.C.D.
`config`	Configuration state.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_CONFIG_INVALID when the network is not NBMA or point-to-multipoint: required for this configuration

OSPF_API_SET_ERR_NBR_NBMA_CONFIG_INVALID when the network is not point-to-multipoint network: required for this configuration

OSPF_API_SET_ERR_NBR_P2MP_CONFIG_REQUIRED when cost option is not configured for the

point-to-multipoint broadcast network

OSPF_API_SET_ERR_NBR_P2MP_CONFIG_INVALID when the network is not NBMA network: required for poll and priority option

ospf_nbr_static_cost_set

This function sets the cost of the specified non-broadcast multi-access (NBMA) neighbor.

This function is called by the following command:

```
neighbor cost
```

Syntax

```
int
ospf_nbr_static_cost_set (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr addr, u_int16_t cost);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address: A.B.C.D.
`cost`	The link state metric to this neighbor: <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_cost_unset

This function resets the cost of the specified non-broadcast multi-access (NBMA) neighbor to the default value: 0.

This function is called by the following command:

```
no neighbor cost
```

Syntax

```
int
ospf_nbr_static_cost_unset (u_int32_t vr_id, int proc_id,
                            struct pal_in4_addr addr);
```

Input Parameter

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address: A.B.C.D.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the proc ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_poll_interval_set

This function sets the poll interval of the non-broadcast multi-access (NBMA) neighbor.

This function is called by the following command:

```
neighbor poll-interval
```

Syntax

```
int
ospf_nbr_static_poll_interval_set (int proc_id, struct pal_in4_addr addr,
                                   int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`interval`	The poll interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

ospf_nbr_static_poll_interval_unset

This function resets the poll interval of the specified non-broadcast multi-access (NBMA) neighbor to the default value specified by `OSPF_POLL_INTERVAL_DEFAULT` (refer to `ospf_interface.h`).

This function is called by the following command:

```
no neighbor poll-interval
```

Syntax

```
int
ospf_nbr_static_poll_interval_unset (u_int32_t vr_id, int proc_id,
                                     struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_priority_set

This function sets the priority of the specified non-broadcast multi-access (NBMA) neighbor.

This function is called by the following command:

```
neighbor priority
```

Syntax

```
int
ospf_nbr_static_priority_set (u_int32_t vr_id, int proc_id,
                              struct pal_in4_addr addr, u_char priority);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`priority`	The neighbor priority <0–255>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_priority_unset

This function resets the priority of the non-broadcast multi-access (NBMA) neighbor to the default value defined in OSPF_NEIGHBOR_PRIORITY_DEFAULT (refer to `ospf_interface.h`).

This function is called by the following command:

```
no neighbor priority
```

Syntax

```
int
ospf_nbr_static_priority_unset (u_int32_t vr_id, int proc_id,
                                struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_set

This function sets the non-broadcast multi-access (NBMA) neighbor.

This function is called by the following command:

```
neighbor
```

Syntax

```
int
ospf_nbr_static_set (u_int32_t vr_id, int proc_id, struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_EXIST when the given neighbor already exists

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_nbr_static_unset

This function deletes the static non-broadcast multi-access (NBMA) neighbor.

This function is called by the following command:

```
no neighbor
```

Syntax

```
int
```

```
ospf_nbr_static_unset (u_int32_t vr_id, int proc_id,
                       struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NBR_STATIC_NOT_EXIST when the given neighbor does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_network_format_set

This function sets the Area ID for the specified address format: decimal or IP address.

This function is called by the following command:

`network area`

Syntax

```
int
ospf_network_format_set (u_int32_t vr_id, int proc_id,
                         struct pal_in4_addr addr, u_char masklen,
                         struct pal_in4_addr area_id, u_char format);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`masklen`	The prefix length.
`area_id`	The area to which the network belongs.
`format`	The format of the IP address: Decimal or IP address notation.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NETWORK_NOT_EXIST when the specified network does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_network_set

This function enables an interface for the OSPF domain.

This function is called by the following command:

```
network area
```

Syntax

```
int
ospf_network_set (u_int32_t vr_id, int proc_id, struct pal_in4_addr addr,
                  u_char masklen, struct pal_in4_addr area_id, s_int16_t id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`masklen`	The prefix length.
`area_id`	The area to which the network belongs.
`id`	The interface instance ID <0–255>; default value is 0.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NETWORK_EXIST when the specified network already exists

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceed the limit

OSPF_API_SET_ERR_IF_INST_ID_CANT_SET if multiple-instance support is not enabled

OSPF_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF_API_SET_ERR_NETWORK_OWNED_BY_ANOTHER_AREA when the given network is already enabled in a different area

OSPF_API_SET_ERR_NETWORK_WITH_ANOTHER_INST_ID_EXIST when the given network is already enabled with a different instance ID

ospf_network_unset

This function deletes the network area configuration directive matched to a specified prefix and area: disables the interface to the OSPF domain.

This function is called by the following command:

```
no network area
```

Syntax

```
int
ospf_network_unset (u_int32_t vr_id, int proc_id, struct pal_in4_addr addr,
                    u_char masklen, struct pal_in4_addr area_id,
                    s_int16_t id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`masklen`	The prefix length.
`area_id`	The area to which the network belongs.
`id`	The interface instance ID <0–255>, default is 0.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NETWORK_NOT_EXIST when the specified network does not exist

OSPF_API_SET_ERR_AREA_ID_NOT_MATCH when the provided area ID does not match the area the network is in

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF_API_SET_ERR_IF_INST_ID_NOT_MATCH when the given instance does not match the configured instance ID

ospf_network_wildmask_set

This function sets the mask representation for the specified network area.

This function is called by the following command:

```
network area
```

Syntax

```
int
ospf_network_wildmask_set (u_int32_t vr_id, int proc_id,
```

```
                        struct pal_in4_addr addr, u_char masklen,
                        struct pal_in4_addr area_id, u_char wild);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The NBMA neighbor IP address.
`masklen`	The prefix length.
`area_id`	The area to which the network belongs.
`wild`	The wildcard mask, including: OSPF_NETWORK_MASK_DEFAULT 0 OSPF_NETWORK_MASK_DECIMAL 1 OSPF_NETWORK_MASK_WILD 2

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_NETWORK_NOT_EXIST when the specified network does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_opaque_area_lsa_set

This function generates area Opaque LSAs.

This function is called by the following command:

```
opaque WORD
```

Syntax

```
int
ospf_opaque_area_lsa_set (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr area_id, u_char type,
                          u_int32_t id, u_char *data, u_int32_t len);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`type`	Opaque type.
`id`	Opaque ID.
`data`	Opaque data.

`len`	Length of opaque data.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area does not exist

ospf_opaque_as_lsa_set

This function generates Autonomous System (AS) area opaque LSAs.

This function is called by the following command:

```
opaque WORD
```

Syntax

```
int
ospf_opaque_as_lsa_set (u_int32_t vr_id, int proc_id, u_char type,
                        u_int32_t id, u_char *data, u_int32_t len);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	Opaque type.
`id`	Opaque ID.
`data`	Opaque data.
`len`	Length of opaque data.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_opaque_data_validate_and_send

Depending on type of opaque LSA information requested, this function validates the Opaque LSA and sends the relevant information to the requesting application. The application can request Opaque LSAs for a specific Opaque LSA type or all Opaque LSA types.

The following information is sent to the requesting application for each Opaque LSA:

- Advertising Router
- Flooding scope
- Opaque Type
- Opaque ID
- Opaque data
- Opaque data length

The validation of the Opaque LSA depends on its flooding scope:

- For Link-local flooding scope, the originator is the OSPF neighbor
- For Area flooding scope, the originator is part of the SPF tree
- For AS-scope flooding scope, a routing table has the entry for Autonomous System Boundary Router (ASBR)

Syntax

```
int
ospf_opaque_data_validate_and_send (struct cli *cli, int proc_id, int type);
```

Input Parameters

`cli`	Pointer to the CLI structure
`proc_id`	The process ID for the OSPF instance <0–65535>
`type`	Opaque LSA type

Output Parameters

None

Return Value

CLI_SUCCESS when the function succeeds

CLI_ERROR when the memory allocation fails

ospf_opaque_link_lsa_set

This function generates the specified AS-wide Opaque LSA.

This function is called by the following command:

```
opaque WORD
```

Syntax

```
int
ospf_opaque_link_lsa_set (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr addr, u_char type,
                          u_int32_t id, u_char *data, u_int32_t len);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	Address.
`type`	Opaque type.
`id`	Opaque ID.
`data`	Opaque data.
`len`	Length of opaque data.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IF_NOT_EXIST when the interface is not found

ospf_overflow_database_external_interval_set

This function sets the value of the time-to-recover interval of the overflow state.

This function is called by the following command:

```
no overflow database external (interval)
```

Syntax

```
int
ospf_overflow_database_external_interval_set (u_int32_t vr_id, int proc_id,
                                              int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`interval`	The time-to-recover interval in seconds.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_overflow_database_external_interval_unset

This function resets the value of the interval of the overflow state as defined by `OSPF_DEFAULT_EXIT_OVERFLOW_INTERVAL` (refer to `ospfd.h`).

This function is called by the following command:

```
no overflow database external
```

Syntax

```
int
ospf_overflow_database_external_interval_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_overflow_database_external_limit_set

This function sets the maximum number of LSAs as specified.

This function is called by the following command:

```
overflow database external (limit)
```

Syntax

```
int
ospf_overflow_database_external_limit_set (u_int32_t vr_id, int proc_id,
                                           u_int32_t limit);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`limit`	The limit.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_overflow_database_external_limit_unset

This function resets the value of the limit of overflow state as defined by `OSPF_DEFAULT_LSDB_LIMIT` (located in `ospfd.h`).

This function is called by the following command:

```
no overflow database external
```

Syntax

```
int
ospf_overflow_database_external_limit_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_process_set

This function creates an OSPF instance, administratively enables it, and puts it in the global OSPF structure. When this function completes, the OSPF instance structure can be accessed with the specified process ID.

Syntax

```
int
ospf_process_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_process_unset

This function destroys the specified OSPF process.

This function is called by the following command:

```
no router ospf
```

Syntax

```
int
ospf_process_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_redistribute_default_set

This function turns on the default originate with the specified origin. This function is called by the following command:

```
redistribute metric-type
```

Syntax

```
int
ospf_redistribute_default_set (u_int32_t vr_id, int proc_id, int origin,
                               int mtype, int mvalue);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`origin`	Default origin, one of these three values: OSPF_DEFAULT_ORIGINATE_UNSPEC OSPF_DEFAULT_ORIGINATE_NSM

	OSPF_DEFAULT_ORIGINATE_ALWAYS
`mtype`	Metric type, routes are redistributed into OSPF as the specified type:
	EXTERNAL_METRIC_TYPE_1
	EXTERNAL_METRIC_TYPE_2
`mvalue`	The value assigned to the route.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_METRIC_INVALID when the `metric value` is outside the range

OSPF_API_SET_ERR_METRIC_TYPE_INVALID when the `type` is not in the list

OSPF_API_SET_ERR_DEFAULT_ORIGIN_INVALID when the `origin` is not in the list

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the protocol is not within range

ospf_redistribute_set

This function turns on redistribute from the specified protocol.

This function is called by the following command:

```
redistribute metric-type
```

Syntax

```
int
ospf_redistribute_set (u_int32_t vr_id, int proc_id,
                       int sproc_id, int type, int mtype, int mvalue);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`sproc_id`	The process ID for the OSPF instance used for redistribution from one OSPF instance to another. <1–65535>.
`type`	Protocol type of route; one of the constants from Protocol Constants
`mtype`	Metric type:
	EXTERNAL_METRIC_TYPE_1
	EXTERNAL_METRIC_TYPE_2
	EXTERNAL_METRIC_TYPE_UNSPEC
`mvalue`	The value assigned to the route.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_METRIC_INVALID when the `metric value` is outside the range

OSPF_API_SET_ERR_METRIC_TYPE_INVALID when the `type` is not in the list

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the `protocol type` is not in the list

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_routemap_set

This function sets the route-map to specified protocol.

This function is called by the following command:

```
redistribute route-map
```

Syntax

```
int
ospf_routemap_set (u_int32_t vr_id, int proc_id, int type, char *name,
int sproc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	Protocol type of route; one of the constants from Protocol Constants
`name`	The name of the route map.
`sproc_id`	The process ID for the OSPF instance used for redistribution from one OSPF instance to another <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the `protocol type` is not in the list

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_routemap_unset

This function resets the route-map to the default protocol.

This function is called by the following command:

```
no redistribute route-map
```

Syntax

```
int
ospf_routemap_unset (u_int32_t vr_id, int proc_id, int type, int sproc_id,);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`type`	Protocol type of route; one of the constants from Protocol Constants
`sproc_id`	The process ID for the OSPF instance used for redistribution from one OSPF instance to another <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the `protocol type` is not in the list

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_router_id_set

This function sets the static OSPF router ID to the specified value.

This function is called by the following command:

```
ospf router-id
```

Syntax

```
int
ospf_router_id_set (u_int32_t vr_id, int proc_id,
                    struct pal_in4_addr router_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`router_id`	The router identifier.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_router_id_unset

This function resets the static OSPF router ID to the default value: 0.

This function is called by the following command:

```
no ospf router_id
```

Syntax

```
int
ospf_router_id_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_set_frr

This function enables or disables fast rerouting on all OSPF interfaces.

This function is called by the following command:

```
fast-reroute keep-all-paths
```

Syntax

```
int32_t
ospf_set_frr (u_int32_t vr_id,
              int proc_id,
              int frr_set
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`proc_id`	The process ID for the OSPF instance <0–65535>.
`frr_set`	One of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Enable fast rerouting.
`PAL_FALSE`	Disable fast rerouting.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when `proc_id` is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when `proc_id` cannot be found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_set_frr_interface

This function permits the interface to be used as the next hop in a repair path or prohibits the interface from being used as the next hop in a repair path.

This function is called by the following command:

```
ip ospf fast-reroute per-prefix candidate disable
```

Syntax

```
int32_t
ospf_set_frr_interface (u_int32_t vr_id,
                        struct interface *ifp,
                        u_int32_t frr_if_set)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`ifp`	Pointer to the interface.
`frr_if_set`	One of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Use `ifp` as the next hop in a repair path.
`PAL_FALSE`	Do not use `ifp` as the next hop in a repair path.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when `proc_id` is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when `proc_id` cannot be found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_set_frr_tie_break_prority

This function sets the priority for a type of fast reroute repair path.

This function is called by the following command:

```
fast-reroute tie-break
```

The default priorities are:

1. PRIMARY_PATH
2. INTERFACE_DISJOINT
3. NODE_PROTECTING
4. BROADCAST_INTERFACE_DISJOINT

Syntax

```
int32_t
ospf_set_frr_tie_break_prority(u_int32_t vr_id,
                               int proc_id,
                               u_int32_t val,
                               u_int32_t index,
                               bool_t isSet)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0.

`proc_id` The process ID for the OSPF instance <0–65535>.

`val` The type of repair path; one of these constants from `ospfd/ospfd.h`:

`PRIMARY_PATH`

Use a path from the Equal-Cost Multipath Path (ECMP) set. An ECMP found during the primary shortest path first (SPF) repair might not be desirable in networks where traffic exceeds the capacity of any single link.

`INTERFACE_DISJOINT`

Do not select point-to-point interfaces that have no alternate next hop for rerouting if the primary gateway fails, thus protecting the interface.

`NODE_PROTECTING`

Bypass the `primary-path` gateway router which might not protect the router that is the next hop in the primary path.

`BROADCAST_INTERFACE_DISJOINT`

Do not use the interface if connected to a broadcast network. Repair paths protect links when a repair path and a protected primary path use *different* next-hop interfaces. However, on broadcast interfaces, if the repair path is computed via the same interface as the primary path, but their next-hop gateways are different, the router is protected but the link might not be.

`index` Tiebreak priority `<1-10>`. A lower value has higher preference.

`isSet` Whether to set the priority for the repair path type specified by `val` to its default value or to a different value; one of these constants from `ospfd/ospf_api.h`:

`OSPF_FRR_TIE_BREAK_SET`

Set the repair path priority to the value specified by `index`

`OSPF_FRR_TIE_BREAK_UNSET`

Set the repair path priority to its default value

Output Parameters

None

Return Value

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when `proc_id` is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when `proc_id` cannot be found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_summary_address_not_advertise_set

This function sets the flag of the external summary address range to `NotAdvertise`.

This function is called by the following command:

```
summary-address not-advertise
```

Syntax

```
int
ospf_summary_address_not_advertise_set (u_int32_t vr_id, int proc_id,
                                        struct pal_in4_addr addr,
                                        u_char masklen);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`proc_id` The process ID for the OSPF instance <0–65535>.

`addr` The network address.

`masklen` The prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the specified summary address does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_summary_address_not_advertise_unset

This function resets the flag of the external summary address range to `OSPF_AREA_RANGE_ADVERTISE`.

This function is called by the following command:

```
no summary-address not-advertise
```

Syntax

```
int
ospf_summary_address_not_advertise_unset (u_int32_t vr_id, int proc_id,
                                          struct pal_in4_addr addr,
                                          u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The network address.
`masklen`	The prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the specified summary address does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_summary_address_set

This function sets the external summary address range.

This function is called by the following command:

```
summary-address
```

Syntax

```
int
ospf_summary_address_set (u_int32_t vr_id, int proc_id,
                          struct pal_in4_addr addr, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The network address.
`masklen`	The prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_EXIST when the specified summary address already exists

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_summary_address_tag_set

This function sets the tag value to the specified value. A tag value that can be used as a “match” value for controlling redistribution via route maps.

This function is called by the following command:

```
summary address tag
```

Syntax

```
int
ospf_summary_address_tag_set (u_int32_t vr_id, int proc_id,
                              struct pal_in4_addr addr, u_char masklen,
                              u_int32_t tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The network address.
`masklen`	The prefix length.
`tag`	The tag value.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the given summary address does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_summary_address_tag_unset

This function resets the tag value of the external summary address range to zero.

This function is called by the following command:

```
no summary address tag
```

Syntax

```
int
ospf_summary_address_tag_unset (u_int32_t vr_id, int proc_id,
                                struct pal_in4_addr addr, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The network address.
`masklen`	The prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the specified summary address does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_summary_address_unset

This function resets the external summary address range.

This function is called by the following command:

```
no summary-address
```

Syntax

```
int
ospf_summary_address_unset (u_int32_t vr_id, int proc_id,
                            struct pal_in4_addr addr, u_char masklen);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`addr`	The network address.
`masklen`	The prefix length.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the specified summary address does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_timers_refresh_set

This function sets the LSA refresh timer value.

This function is called by the following command:

```
refresh timer
```

Syntax

```
int
ospf_timers_refresh_set (u_int32_t vr_id, int proc_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`interval`	The refresh timer interval in seconds.<10-1800>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_TIMER_VALUE_INVALID when the interval value is outside the range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_timers_refresh_unset

This function resets the LSA refresh timer to the default value (OSPF_LSA_REFRESH_INTERVAL_DEFAULT).

This function is called by the following command:

```
no refresh timer
```

Syntax

```
int
ospf_timers_refresh_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_timers_spf_set

This function sets the minimum and maximum delay between a topology change, being either received in an LSA or self detected, and the SPF calculation being run.

- Delay time between OSPF receives a topology change and it starts an SPF calculation

This function is called by the following command:

```
timers spf exp
```

Syntax

```
int
ospf_timers_spf_set (u_int32_t vr_id, int proc_id,
                     u_int32_t min_delay, u_int32_t max_delay);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`min_delay`	The minimum SPF hold delay time in milliseconds: 500 milliseconds.
`max_delay`	The maximum SPF hold delay time in milliseconds: 50000 milliseconds (50 seconds).

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_timers_spf_unset

This function resets the SPF minimum delay and maximum delay to their default values: 5 seconds.

This function is called by the following command:

```
no timers spf exp
```

Syntax

```
int
ospf_timers_spf_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_unset_frr_tie_break_prority_all

This function sets the priority for all types of fast reroute repair paths to their default values.

This function is called by the following command:

```
no fast-reroute tie-break
```

The default priorities are:

1. PRIMARY_PATH
2. INTERFACE_DISJOINT
3. NODE_PROTECTING
4. BROADCAST_INTERFACE_DISJOINT

Syntax

```
int32_t
ospf_unset_frr_tie_break_prority_all(u_int32_t vr_id, int proc_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_authentication_key_set

This function sets the simple authentication password (type 1) for the OSPF virtual links.

- Simple password authentication allows a password (key) to be configured per area.
- To participate in the routing domain, routers in the same area must be configured with the same key.

This function is called by the following command:

```
area virtual-link authentication-key
```

Syntax

```
int
ospf_vlink_authentication_key_set (u_int32_t vr_id, int proc_id,
                                   struct pal_in4_addr area_id,
                                   struct pal_in4_addr peer_id,
                                   char *authkey);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs: IP address format.
`peer_id`	Neighbor Router ID.
`authkey`	The password to be used by neighbors: maximum eight characters.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_authentication_key_unset

This function resets the simple authentication password for the virtual link to NULL (0).

This function is called by the following command:

```
no area virtual-link authentication-key
```

Syntax

```
int
ospf_vlink_authentication_key_unset (u_int32_t vr_id, int proc_id,
                                     struct pal_in4_addr area_id,
                                     struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified parameters are not configured

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_authentication_type_set

This function sets the authentication type for the virtual interface.

This function is called by the following command:

```
area virtual-link authentication
```

Syntax

```
int
ospf_vlink_authentication_type_set (u_int32_t vr_id, int proc_id,
                                    struct pal_in4_addr area_id,
                                    struct pal_in4_addr peer_id, int type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

area_id	OSPF are ID in IP address format.
peer_id	Neighbor Router ID, IP address of the virtual link neighbor.
type	Authentication type: Null Simple password Cryptographic

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AUTH_TYPE_INVALID when the given authentication type is not within the specified list

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_authentication_type_unset

This function resets the authentication type for the virtual interface to NULL (0).

This function is called by the following command:

```
no area virtual-link authentication
```

Syntax

```
int
ospf_vlink_authentication_type_unset (u_int32_t vr_id, int proc_id,
                                      struct pal_in4_addr area_id,
                                      struct pal_in4_addr peer_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	The area to which the network belongs.
peer_id	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified interface parameters are not configured

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_dead_interval_set

This function sets the router dead interval value for the virtual interface.

This function is called by the following command:

```
area virtual-link dead-interval
```

Syntax

```
int
ospf_vlink_dead_interval_set (u_int32_t vr_id, int proc_id,
                              struct pal_in4_addr area_id,
                              struct pal_in4_addr peer_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.
`interval`	The dead interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_IF_DEAD_INTERVAL_NOT_VALID when the specified dead interval is out of range

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_dead_interval_unset

This function resets the router dead interval value for the virtual interface to the default value: 40.

This function is called by the following command:

```
no area virtual-link dead-interval
```

Syntax

```
int
ospf_vlink_dead_interval_unset (u_int32_t vr_id, int proc_id,
                                struct pal_in4_addr area_id,
```



```
                                struct pal_in4_addr peer_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	The area to which the network belongs.
peer_id	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified interface parameters are not configured

ospf_vlink_format_set

This function specifies the format of the virtual link IP address, either decimal or A.B.C.D.

This function is called by the following command:

```
area <area_id> virtual-link<peer_id>
```

Syntax

```
int
ospf_vlink_format_set (u_int32_t vr_id, int proc_id,
                       struct pal_in4_addr area_id,
                       struct pal_in4_addr peer_id, u_char format);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
proc_id	The process ID for the OSPF instance <0–65535>.
area_id	OSPF area ID in IP address format.
peer_id	Neighbor Router ID.
format	The format of the address: OSPF_AREA_ID_FORMAT_ADDRESS OSPF_AREA_ID_FORMAT_DECIMAL

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_ID_FORMAT_INVALID when the format is not supported

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_hello_interval_set

This function sets the router hello interval value for the virtual interface. The hello interval value must be the same for both ends of the virtual link.

This function is called by the following command:

```
virtual-link hello-interval
```

Syntax

```
int
ospf_vlink_hello_interval_set (u_int32_t vr_id, int proc_id,
                               struct pal_in4_addr area_id,
                               struct pal_in4_addr peer_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID in IP address format.
`peer_id`	Neighbor Router ID.
`interval`	The hello interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when the value is outside the range

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_hello_interval_unset

This function resets the router hello interval value for the virtual interface to the default value: 10 seconds.

This function is called by the following command:

```
no area virtual-link hello-interval
```

Syntax

```
int
ospf_vlink_hello_interval_unset (u_int32_t vr_id, int proc_id,
                                 struct pal_in4_addr area_id,
                                 struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	OSPF area ID in IP address format.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified interface parameters are not configured

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_message_digest_key_set

This function sets the MD5 authentication key for the virtual interface.

This function is called by the following command:

```
area virtual-link message-digest-key
```

Syntax

```
int
ospf_vlink_message_digest_key_set (u_int32_t vr_id, int proc_id,
                                   struct pal_in4_addr area_id,
                                   struct pal_in4_addr peer_id,
                                   u_char key_id, char *auth_key);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.
`key_id`	The key identifier.

`auth_key`	The message digest key string.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_MD5_KEY_EXIST when the given key already exists

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_message_digest_key_unset

This function deletes the MD5 authentication key for the virtual interface.

This function is called by the following command:

```
no area virtual-link message-digest-key
```

Syntax

```
int
ospf_vlink_message_digest_key_unset (u_int32_t vr_id, int proc_id,
                                     struct pal_in4_addr area_id,
                                     struct pal_in4_addr peer_id,
                                     u_char key_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.
`key_id`	The key identifier.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified interface parameters are not configured

OSPF_API_SET_ERR_MD5_KEY_NOT_EXIST when the specified key does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_retransmit_interval_set

This function sets the retransmit interval value for the virtual interface.

This function is called by the following command:

```
virtual-link retransmit-interval
```

Syntax

```
int
ospf_vlink_retransmit_interval_set (u_int32_t vr_id, int proc_id,
                                    struct pal_in4_addr area_id,
                                    struct pal_in4_addr peer_id,
                                    int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.
`interval`	The retransmit interval in seconds <5–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when the value is outside the range

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_retransmit_interval_unset

This function resets the retransmit interval value for the virtual interface to the default value: 5 seconds.

This function is called by the following command:

```
no virtual-link retransmit-interval
```

Syntax

```
int
ospf_vlink_retransmit_interval_unset (u_int32_t vr_id, int proc_id,
                                      struct pal_in4_addr area_id,
                                      struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the specified interface parameters are not configured

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_set

This function creates a virtual interface and configures a virtual neighbor.

This function is called by the following command:

```
area virtual-link
```

Note: For the multi-area adjacency feature, this function checks to determine whether the area is configured for multi-area adjacency. If configured, this function returns an error.

Syntax

```
int
ospf_vlink_set (u_int32_t vr_id, int proc_id, struct pal_in4_addr area_id,
                struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_CANT_GET when the specified virtual link cannot be obtained

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_AREA_LIMIT when the number of areas exceeds the limit

OSPF_API_SET_ERR_AREA_NOT_DEFAULT when the specified area is not the default area

ospf_vlink_transmit_delay_set

This function sets the specified transmit delay (in seconds) for the virtual link.

This function is called by the following command:

```
area virtual-link transmit-delay
```

Syntax

```
int
ospf_vlink_transmit_delay_set (u_int32_t vr_id, int proc_id,
                               struct pal_in4_addr area_id,
                               struct pal_in4_addr peer_id, int delay);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.
`delay`	The transmit delay in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when the value is outside the range

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_transmit_delay_unset

This function resets the transmit delay for the interface: 1 seconds.

This function is called by the following command:

```
no area virtual-link transmit-delay
```

Syntax

```
int
ospf_vlink_transmit_delay_unset (u_int32_t vr_id, int proc_id,
                                 struct pal_in4_addr area_id,
```

```
                                    struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameters are not configured

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_vlink_unset

This function destroys the specified virtual interface and deconfigures the specified virtual neighbor.

This function is called by the following command:

```
no area virtual-link
```

Syntax

```
int
ospf_vlink_unset (u_int32_t vr_id, int proc_id, struct pal_in4_addr area_id,
                  struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`area_id`	The area to which the network belongs.
`peer_id`	Neighbor Router ID.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is not valid (outside the 1–65535 bounds)

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VLINK_NOT_EXIST when the given virtual link does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPFv2 Graceful Restart API

The following is a summary of the OSPFv2 graceful restart API functions. Details are provided in the following subsections. For information about the OSPF graceful restart feature, see Chapter 2, *Graceful Restart*.

Function	Description
ospf_capability_restart_set	Enables OSPF graceful restart capability
ospf_capability_restart_unset	Disables the OSPF graceful restart capability
ospf_graceful_restart_planned_set	Configures only OSPF planned (S/W) restarts.
ospf_graceful_restart_planned_unset	Configures all types of restarts (Planned & Unplanned).
ospf_graceful_restart_set	Configures the grace period (in seconds) for restarting the router
ospf_graceful_restart_unset	Disables graceful restart
ospf_process_unset_graceful	Forces the shutdown of all existing OSPF instances as graceful restart
ospf_restart_graceful	Restarts OSPF gracefully with the given grace period and reason for restart
ospf_restart_helper_grace_period_set	Sets the maximum grace period allowed to be helper for restarting a router
ospf_restart_helper_grace_period_unset	Resets the maximum grace period to default
ospf_routemap_set	Sets graceful restart helper behavior for the specified router
ospf_restart_helper_never_router_unset	Removes the specified neighbor ID from the router ID list
ospf_restart_helper_never_router_unset_all	Removes all neighbor IDs from the router ID list
ospf_restart_helper_policy_set	Sets helper behavior for OSPF graceful restart
ospf_restart_helper_policy_unset	Resets helper to “always accept” (default)

Include Files

To call the functions in this section, include `ospfd/ospf_api.h`.

ospf_capability_restart_set

This function enables OSPF graceful restart capability.

This function is called by the following command:

```
capability restart
```

Syntax

```
int
ospf_capability_restart_set (u_int32_t vr_id, int proc_id, int method);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`method`	Graceful restart or restart signalling.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_RESTART_METHOD_INVALID when the restart method is invalid

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_capability_restart_unset

This function disables the OSPF graceful restart capability.

This function is called by the following command:

```
no capability restart
```

Syntax

```
int
ospf_capability_restart_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_graceful_restart_planned_set

This function allows to configure only OSPF planned (S/W) restarts.

This function is called by the following command:

```
restart ospf hitless (grace-period <1-1800>|)
```

Syntax

```
int
ospf_graceful_restart_planned_set (struct ospf_master *om);
```

Input Parameters

`om`	Pointer to the Master OSPF structure

Output Parameters

None

Return Value

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_graceful_restart_planned_unset

This function allows to configure all types of restarts (Planned & Unplanned).

This function is called by the following command:

```
no graceful-restart ospf planned-only
```

Syntax

```
int
ospf_graceful_restart_planned_unset (struct ospf_master *om);
```

Input Parameters

`om`	Pointer to the Master OSPF structure

Output Parameters

None

Return Value

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_graceful_restart_set

This function sets the grace period, this value is used to restart OSPF gracefully when there is an unexpected shutdown.

This function is called by the following command:

`ospf restart grace-period` <1-1800>

Syntax

```
int
ospf_graceful_restart_set (struct ospf_master *om, int seconds, int reason);
```

Input Parameters

`om`	Pointer to the Master OSPF structure
`seconds`	Grace period in seconds for graceful restart <1–1800>

`reason` Restart reason; one of the constants from `ospfd/ospfd.h`:

- `OSPF_RESTART_REASON_UNKNOWN`
- `OSPF_RESTART_REASON_RESTART`
- `OSPF_RESTART_REASON_UPGRADE`
- `OSPF_RESTART_REASON_SWITCH_REDUNDANT`

Output Parameters

None

Return Value

OSPF_API_SET_ERR_GRACE_PERIOD_INVALID when given value is outside of the range

OSPF_API_SET_ERR_INVALID_REASON when given value is outside of the range

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_graceful_restart_unset

This function disables graceful restart: resets the grace period to 0.

This function is called by the following command:

```
no ospf restart grace-period
```

Syntax

```
int
ospf_graceful_restart_unset (struct ospf_master *om);
```

Input Parameters

`om` Pointer to the Master OSPF structure

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_process_unset_graceful

This function forces the shutdown of all existing OSPF instances as Graceful Restart.

This function is called by the following command:

```
restart ospf graceful
```

Note: For the multi-area adjacency feature, this function sends grace LSAs for multi-area networks.

Syntax

```
int
```



```
ospf_process_unset_graceful (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the given process ID is outside of the range

OSPF_API_SET_ERR_PROCESS_NOT_EXIST if process as given process ID does not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf_restart_graceful

This function restarts OSPF gracefully with the given grace period along with the reason for restart

This function is called by the following command

```
restart ospf graceful(grace-period <1-1800>|)
```

Syntax

```
int
ospf_restart_graceful_sdkapi (u_int16_t grace_period,
        int reason);
```

Input Parameters

`grace period`	Grace-period in seconds <1-1800>
`reason`	Reason for restart
	`OSPF_RESTART_REASON_UNKNOWN`
	`OSPF_RESTART_REASON_RESTART`
	`OSPF_RESTART_REASON_UPGRADE`
	`OSPF_RESTART_REASON_SWITCH_REDUNDANT`

Output Parameters

None

Return Value

OSPF_API_SET_ERR_GRACE_PERIOD_INVALID when given value is outside of the range

OSPF_API_SET_ERR_INVALID_REASON when given value is outside of the range

OSPF_API_SET_MALLOC_ERR when there is an error allocating memory

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_ID_INVALID hen the given process ID is outside of the range

OSPF_API_SET_ERR_GR_NOT_SUPPORTED when OSPF SSO is enabled

ospf_restart_helper_grace_period_set

This function sets the maximum grace period allowed to be helper for restarting a router.

This function is called by the following command:

```
ospf graceful-restart helper max-grace-period
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`seconds`	Maximum grace period to accept.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_GRACE_PERIOD_INVALID when given value is outside of the range

OSPF_API_SET_SUCCESS when the function succeeds

ospf_restart_helper_grace_period_unset

This function resets the maximum grace period.

This function is called by the following command:

```
no ospf graceful-restart helper max-grace-period
```

Syntax

```
int
ospf_restart_helper_grace_period_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

ospf_restart_helper_never_router_set

This function sets graceful restart helper behavior for the specified router.

This function is called by the following command:


```
ospf restart helper never router-id
```

Syntax

```
int
ospf_restart_helper_never_router_set (u_int32_t vr_id,
                                      struct pal_in4_addr router_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`router-id`	Router ID of neighbor to never to act as helper.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_IP_ADDR_IN_USE when the specified router ID is already configured

OSPF_API_SET_ERR_MALLOC_FAIL_FOR_ROUTERID when the memory allocation fails for the new router ID

ospf_restart_helper_never_router_unset

This function removes the specified neighbor ID from the router ID list.

This function is called by the following command:

```
no ospf restart helper never router-id
```

Syntax

```
int
ospf_restart_helper_never_router_unset (u_int32_t vr_id,
                                        struct pal_in4_addr router_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`router-id`	Router ID of neighbor to never to act as helper.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_restart_helper_never_router_unset_all

This function removes all neighbor IDs from the router ID list.

This function is called by the following command:

```
no ospf restart helper never router-id all
```

Syntax

```
int
ospf_restart_helper_never_router_unset_all (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_EMPTY_NEVER_RTR_ID when the specified router ID does not exist

ospf_restart_helper_policy_set

This function sets helper behavior for OSPF graceful restart.

This function is called by the following command:

```
spf graceful-restart helper
```

Syntax

```
int
ospf_restart_helper_policy_set (u_int32_t vr_id, int policy);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`policy`	Helper Policy, one of the following:
`1`	Helper never using the OSPF_RESTART_HELPER_NEVER define
`2`	Helper only during reload using the OSPF_RESTART_HELPER_ONLY_RELOAD define
`3`	Helper only during upgrade using the OSPF_RESTART_HELPER_ONLY_UPGRADE

Output Parameters

None

Return Value

OSPF_API_SET_ERR_INVALID_HELPER_POLICY when the given policy is outside of the range

OSPF_API_SET_SUCCESS when the function succeeds

ospf_restart_helper_policy_unset

This function resets helper to the default behavior: always accept.

This function is called by the following command:

```
no ospf graceful-restart helper
```

Syntax

```
int
ospf_restart_helper_policy_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPFv2 Passive-Interface API

The following is a summary of the OSPFv2 API that support the passive interface feature. Details are provided in the following subsections. For more information about the passive interface feature, see Chapter 6, *Passive Interface*

.

Function	Description
ospf_passive_interface_default_set	Sets all interfaces to passive mode
ospf_passive_interface_default_unset	Resets all interfaces to active mode (default setting)
ospf_passive_interface_set	Sets the specified interface to passive mode
ospf_passive_interface_set_by_addr	Sets the interface specified by IP address to passive mode
ospf_passive_interface_unset	Resets the current interface to active mode (default mode)
ospf_passive_interface_unset_by_addr	Resets the interface specified by IP address to active mode

Include File

To call the functions in this section, include `ospfd\ospf_api.h`.

ospf_passive_interface_default_set

This function sets all interfaces to passive mode by default.

This function is called by the following command:

```
ospf passive-interface
```

Syntax

```
int
```

```
ospf_passive_interface_default_set (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_passive_interface_default_unset

This function resets all interfaces.

This function is called by the following command:

```
no ospf passive interface
```

Syntax

```
int
ospf_passive_interface_default_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_passive_interface_set

This function sets the specified interface to passive mode (OSPF_IF_PASSIVE).

This function is called by the following command:

```
passive interface
```

Syntax

```
int
ospf_passive_interface_set (u_int32_t vr_id, int proc_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_passive_interface_set_by_addr

This function sets the interface specified by IP address to passive mode (OSPF_IF_PASSIVE).

This function is called by the following command:

```
passive interface
```

Syntax

```
int
ospf_passive_interface_set_by_addr (u_int32_t vr_id, int proc_id,
                                    char *name, struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>.
`name`	The interface name.
`addr`	The IP address of the interface.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_passive_interface_unset

This function resets the current interface to active mode (default setting, OSPF_IF_ACTIVE).

This function is called by the following command:

```
no passive interface
```

Syntax

```
int
ospf_passive_interface_unset (u_int32_t vr_id, int proc_id, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`name`	The interface name.

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf_passive_interface_unset_by_addr

This function resets the interface specified by IP address to active mode (OSPF_IF_ACTIVE).

This function is called by the following command:

```
no passive interface
```

Syntax

```
int
ospf_passive_interface_unset_by_addr (u_int32_t vr_id, int proc_id,
                                      char *name, struct pal_in4_addr addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	The process ID for the OSPF instance <0–65535>
`name`	The interface name
`addr`	The IP address of the interface

Output Parameters

None

Return Value

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

CHAPTER 13 OSPFv3 Command API

This chapter describes the OSPFv3 Command API. The following sections are covered in this section:

- OSPFv3 API
- OSPFv3 Graceful Restart API

OSPFv3 API

The following is a summary of the OSPFv3 functions. Details are provided in the following subsections.

Function	Description
ospf6_address_family_set	Sets address family mode to IPv4 unicast or IPv6 unicast
ospf6_abr_type_set	Sets the ABR type
ospf6_abr_type_unset	Sets the ABR type to Cisco
ospf6_area_default_cost_set	Configures the stub default cost
ospf6_area_default_cost_unset	Unsets the stub default cost
ospf6_area_format_set	Sets the format of the Area ID
ospf6_area_no_summary_set	Configures an area as stub no-summary
ospf6_area_no_summary_unset	Unconfigures the area as stub no-summary
ospf6_area_nssa_default_originate_metric_set	Originates Type-7 default LSAs with a specific metric into an NSSA area
ospf6_area_nssa_default_originate_metric_type_set	Originates Type-7 default LSAs with a specific metric type into an NSSA area
ospf6_area_nssa_default_originate_set	Originates Type-7 default LSAs into an NSSA area
ospf6_area_nssa_default_originate_unset	Stops originating Type-7 default LSAs into an NSSA area
ospf6_area_nssa_no_redistribution_set	Stops redistributing routes into an NSSA area
ospf6_area_nssa_no_redistribution_unset	Redistributes routes into an NSSA area
ospf6_area_nssa_set	Sets an area as an NSSA
ospf6_area_nssa_stability_interval_set	Sets the stability interval for an NSSA area
ospf6_area_nssa_translator_role_set	Sets the translator role for an NSSA area
ospf6_area_nssa_translator_role_unset	Removes the translator role for an NSSA area
ospf6_area_nssa_unset	Makes an area a normal area

Function	Description
ospf6_area_range_set	Configures an area range for an IPv6 prefix
ospf6_area_range_unset	Unsets the configured area range
ospf6_area_stub_set	Configures the specified OSPFv3 area as stub
ospf6_area_stub_unset	Makes an area a normal area
ospf6_auto_cost_reference_bandwidth_set	Sets the reference bandwidth value
ospf6_auto_cost_reference_bandwidth_unset	Sets the reference bandwidth to its default value (100000 Kbps)
ospf6_capability_cspf_set	Enables the CSPF capability for an OSPFv3 process
ospf6_capability_cspf_unset	Unsets the CSPF capability for an OSPFv3 process
ospf6_default_metric_set	Sets the specified default metric for external routes
ospf6_default_metric_unset	Unsets the configured default metric
ospf6_disable_db_summary_opt	Disables the OSPFv3 Database Summary List optimization
ospf6_dna_set	Enables the OSPFv3 flood-reduction
ospf6_dna_unset	Disables the OSPFv3 flood-reduction
ospf6_frr_set	Enables the OSPFv3 Loop-free alternate
ospf6_frr_interface_set	Disables the OSPFv3 LFA calculation on interface
ospf6_frr_tie_break_prority_set	Enables the OSPFv3 tie-breaker attribute for LFA
ospf6_frr_tie_break_priority_all_unset	Disables all the configure tie-breaking priorities
ospf6_if_cost_set	Resets the grace period to the default value
ospf6_if_cost_unset	Unsets the configured output cost on the specified interface
ospf6_if_dc_set	Sets the demand-circuit capability on the specified interface
ospf6_if_dc_unset	Unsets the demand-circuit capability on the specified interface
ospf6_if_dead_interval_set	Configures the Router Dead Interval on a specified interface
ospf6_if_dead_interval_unset	Unconfigures the Router Dead Interval of the specified interface
ospf6_if_dna_set	Enables OSPFv3 flood-reduction on a specified interface
ospf6_if_dna_unset	Disables OSPFV3 flood-reduction on a specified interface
ospf6_if_hello_interval_set	Configures Hello Interval on the specified interface
ospf6_if_hello_interval_unset	Unsets the configured Hello interval on a specified interface
ospf6_if_neighbor_set	Configures the OSPFv3 routers interconnecting to non-broadcast networks

Function	Description
ospf6_if_neighbor_unset	Unconfigures the OSPFv3 routers interconnecting to non-broadcast networks
ospf6_if_network_set	Sets the network type
ospf6_if_network_unset	Removes the network type
ospf6_if_ipv6_router_set	Enables OSPF routing on the specified interface
ospf6_if_ipv6_router_unset	Disables OSPF routing on a specified interface
ospf6_if_link_lsa_suppression_set	Enables or disables link LSA (type 8) suppression
ospf6_if_priority_set	Configures router priority on the specified interface
ospf6_if_priority_unset	Unsets the configured Router Priority of the specified interface
ospf6_if_retransmit_interval_set	Configures the retransmit interval of the specified interface
ospf6_if_retransmit_interval_unset	Unsets the configured retransmit interval of the specified interface
ospf6_if_te_metric_set	Sets the specified metric for OSPFv3 traffic engineering on an interface
ospf6_if_te_metric_unset	Unsets a metric for OSPFv3 traffic engineering on an interface
ospf6_if_transmit_delay_set	Configures transmit delay on a specified interface
ospf6_if_transmit_delay_unset	Unsets the configured transmit delay of the specified interface
ospf6_ipv6_ospf_display_route_single_line_set	Shows the results of `show ipv6 ospf route` in a single line
ospf6_ipv6_ospf_display_route_single_line_unset	Shows the results of `show ipv6 ospf route` in multiple lines
ospf6_max_concurrent_dd_set	Sets the maximum number of concurrently processed database descriptors
ospf6_max_concurrent_dd_unset	Sets the maximum concurrent database descriptors to its default (5)
ospf6_passive_if_set	Configures the specified interface as passive
ospf6_passive_if_unset	Unsets the passive interface configuration
ospf6_redistribute_metric_set	Sets the metric for a redistributed route
ospf6_redistribute_metric_type_set	Sets the metric type for a redistributed route
ospf6_redistribute_metric_type_unset	Sets the metric type of a redistributed route to its default (2)
ospf6_redistribute_metric_unset	Sets the metric of a redistributed route to its default (16777215)
ospf6_redistribute_set	Redistributes routes from the specified protocol
ospf6_redistribute_unset	Stops redistributing routes from the specified protocol
ospf6_routemap_set	Assigns a route-map to a redistributed protocol

Function	Description
ospf6_routemap_unset	Removes a route map from a redistributed protocol
ospf6_router_id_set	Unsets the metric type to the specified route type with redistribution
ospf6_router_id_unset	Unsets the configured OSPFv3 Router ID
ospf6_router_set	Initiates the OSPFv3 router instance
ospf6_router_unset	Removes the OSPFv3 router instance
ospf6_summary_address_not_advertise_set	Suppresses external routes that match a specified address range
ospf6_summary_address_not_advertise_unset	Stops suppressing external routes that match a specified address range
ospf6_summary_address_set	Summarizes external routes with the specified address range
ospf6_summary_address_tag_set	Sets a tag value to use as a “match” value for controlling redistribution via route maps
ospf6_summary_address_tag_unset	Removes a tag value to use as a “match” value for controlling redistribution via route maps
ospf6_summary_address_unset	Removes a summary address
ospf6_timers_spf_set	Sets the minimum and maximum delay between a topology change
ospf6_timers_spf_unset	Resets the SPF minimum and maximum delay to their default values
ospf6_vlink_dead_interval_set	Configures the Router Dead Interval of the specified virtual link
ospf6_vlink_dead_interval_unset	Unsets the configured Router Dead Interval of the specified virtual link
ospf6_vlink_format_set	Specifies the format for the specified virtual link configuration
ospf6_vlink_hello_interval_set	Configures Hello Interval on the specified virtual link
ospf6_vlink_hello_interval_unset	Unsets the configured Hello Interval on a specified virtual link
ospf6_vlink_instance_id_set	Configures the instance ID on the specified virtual link
ospf6_vlink_instance_id_unset	Unsets the configured instance ID of the specified virtual link
ospf6_vlink_retransmit_interval_set	Sets the retransmit interval on the specified virtual link
ospf6_vlink_retransmit_interval_unset	Unsets the configured retransmit interval of the specified virtual link
ospf6_vlink_set	Configures the virtual link as specified
ospf6_vlink_transmit_delay_set	Configures the transmit delay on a specified virtual link.
ospf6_vlink_transmit_delay_unset	Unsets the configured Transmit Delay on specified virtual-link
ospf6_vlink_unset	Unsets the virtual link configuration

Include File

To call the functions in this section, include `ospf6d\ospf6_api.h`.

ospf6_address_family_set

This function sets address family mode to IPv4 unicast or IPv6 unicast.

This function is called by the following commands:

```
address-family ipv4 unicast
no address-family
```

Syntax

```
int
ospf6_address_family_set (u_int32_t vr_id, char *tag, u_char add_family)
```

Input Parameters

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

`tag` OSPFv3 process tag

`add_family` Address family; one of these constants from `ospf6d/ospf6d.h`:

`AF_IPV4_UNICAST`
IPv4 unicast

`AF_IPV6_UNICAST`
IPv6 unicast

Output Parameters

None

Return Values

OSP6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_abr_type_set

This function sets the area border route (ABR) type.

This function is called by the following command:

```
ospf6 abr-type
```

Syntax

```
int
ospf6_abr_type_set (u_int32_t vr_id, char *tag, u_char abr_type);
```

Input Parameters

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

`tag`	OSPFv3 process tag.
`abr_type`	The type of area border router:
`cisco`	Specifies an alternative ABR using the Cisco implementation
`ibm`	Specifies an alternative ABR using the IBM implementation
`standard`	Specifies standard behavior (RFC 2328)

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSP6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_ABR_TYPE_INVALID when the ABR type is not valid

ospf6_abr_type_unset

This function sets the ABR type to Cisco.

This function is called by the following command:

```
no ospf6 abr-type
```

Syntax

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

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF_API_SET_SUCCESS when the function succeeds

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the process is not found

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_area_default_cost_set

This function configures the cost for default summary route sent into a stub area.

This function is called by the following command:

```
area default-cost
```

Syntax

```
int
ospf6_area_default_cost_set (u_int32_t vr_id, char *tag,
                             struct pal_in4_addr area_id, int cost);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
tag	OSPFv3 process tag.
area_id	OSPFv3 area ID in an IPv4 address format.
cost	Stub advertised; default summary cost <0–16777215>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is a Backbone area

OSPF6_API_SET_ERR_METRIC_INVALID when the given cost is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when specified area is default

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_default_cost_unset

This function removes the assigned stub default cost.

This function is called by the following command:

```
no area default-cost
```

Syntax

```
int
ospf6_area_default_cost_unset (u_int32_t vr_id, char *tag,
                               struct pal_in4_addr area_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
tag	OSPFv3 process tag.
area_id	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when specified area is default

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_format_set

This function sets the format of the area ID.

This function is called by the following command:

```
area stub
```

Syntax

```
int
ospf6_area_format_set (u_int32_t vr_id, char *tag,
                       struct pal_in4_addr area_id, u_char format);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag.

`area_id` OSPFv3 area ID in an IPv4 address format.

`format` Area ID format; one of these constants from `lib/ospf_common/ospf_const.h`:

`OSPF_AREA_ID_FORMAT_DEFAULT`
Default format

`OSPF_AREA_ID_FORMAT_ADDRESS`
IP address forma

`OSPF_AREA_ID_FORMAT_DECIMAL`
Decimal format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_ID_FORMAT_INVALID when given area ID format is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_no_summary_set

This function configures an area as stub no-summary: inter-area routes are not injected into stub.

This function is called by the following command:

```
area no-summary
```

Syntax

```
int
ospf6_area_no_summary_set (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when specified area is default

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_no_summary_unset

This function unconfigures the area as stub no-summary.

This function is called by the following commands:

```
area (A.B.C.D|<0-4294967295>) stub
no area (A.B.C.D|<0-4294967295>) stub no-summary
no area (A.B.C.D|<0-4294967295>) nssa no-summary
```

Syntax

```
int
ospf6_area_no_summary_unset (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when specified area is default

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_default_originate_metric_set

This function originates Type-7 default LSAs with a specific metric into an NSSA area.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa default-information-originate metric
```

Syntax

```
int
ospf6_area_nssa_default_originate_metric_set (u_int32_t vr_id, char *tag,
                                              struct pal_in4_addr area_id,
                                              int metric)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format
`metric`	Metric value for default routes <0-16777214>

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_METRIC_INVALID when the specified metric is not within range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_default_originate_metric_type_set

This function originates Type-7 default LSAs with a specific metric type into an NSSA area.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa default-information-originate metric-type
```

Syntax

```
int
ospf6_area_nssa_default_originate_metric_type_set (u_int32_t vr_id, char *tag,
                                                   struct pal_in4_addr area_id,
```

```
                                                int mtype)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format
`mtype`	Metric type (see RFC 3101) <1-2>

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_METRIC_TYPE_INVALID when the specified metric type is not valid

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_default_originate_set

This function originates Type-7 default LSAs into an NSSA area.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa default-information-originate
```

Syntax

```
int
ospf6_area_nssa_default_originate_set (u_int32_t vr_id, char *tag,
                                       struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_default_originate_unset

This function stops originating Type-7 default LSAs into an NSSA area.

This function is called by the following command:

```
no area (A.B.C.D|<0-4294967295>) nssa default-information-originate
```

Syntax

```
int
ospf6_area_nssa_default_originate_unset (u_int32_t vr_id, char *tag,
                                         struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_no_redistribution_set

This function stops redistributing routes into an NSSA area.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa no-redistribution
```

Syntax

```
int
ospf6_area_nssa_no_redistribution_set (u_int32_t vr_id, char *tag,
                                       struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_no_redistribution_unset

This function redistributes routes into an NSSA area.

This function is called by the following command:

```
no area (A.B.C.D|<0-4294967295>) nssa no-redistribution
```

Syntax

```
int
ospf6_area_nssa_no_redistribution_unset (u_int32_t vr_id, char *tag,
                                         struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_set

This function sets an area as an NSSA.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa
```

Syntax

```
int
ospf6_area_nssa_set (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_HAS_VLINK when the specified area has a virtual link

OSPF6_API_SET_ERR_AREA_IS_STUB when the specified area is a stub

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_stability_interval_set

This function sets the stability interval for an NSSA area. If an elected translator determines its services are no longer required, it continues to perform its duties for this time interval. This minimizes excess flushing of translated Type-7 LSAs and provides a more stable translator transition.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa stability-interval <0-2147483647>
```

Syntax

```
int
ospf6_area_nssa_stability_interval_set (u_int32_t vr_id, char *tag,
                                        struct pal_in4_addr area_id, u_int32_t intvl)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format
`intvl`	Stability interval in seconds <0-2147483647>

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_translator_role_set

This function sets the translator role for an NSSA area.

This function is called by the following command:

```
area (A.B.C.D|<0-4294967295>) nssa translator-role (candidate|always)
```

Syntax

```
int
ospf6_area_nssa_translator_role_set (u_int32_t vr_id, char *tag,
                                     struct pal_in4_addr area_id, u_char role)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag.

`area_id` OSPFv3 area ID in an IPv4 address format

`role` NSSA-ABR translator role; one of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATE_CANDIDATE`
Translate NSSA-LSA to Type-5 LSA if router is elected

`OSPF_NSSA_TRANSLATE_ALWAYS`
Always translate NSSA-LSA to Type-5 LSA

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_translator_role_unset

This function removes the translator role for an NSSA area.

This function is called by the following command:

```
no area (A.B.C.D|<0-4294967295>) nssa translator-role
```

Syntax

```
int
ospf6_area_nssa_translator_role_unset (u_int32_t vr_id, char *tag,
                                       struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_nssa_unset

This function makes an area a normal area.

This function is called by the following command:

```
no area (A.B.C.D|<0-4294967295>) nssa
```

Syntax

```
int
ospf6_area_nssa_unset (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the specified area does not exist

OSPF6_API_SET_ERR_AREA_IS_STUB when the specified area is a stub

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_range_set

This function configures an OSPF address range.

This function is called by the following command:

```
area range
```

Syntax

```
int
ospf6_area_range_set (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id,
                      void *addr, int prefixlen, int status
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`area_id`	OSPFv3 area ID in an IPv4 address format
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`prefixlen`	Area range IPv6 prefix length <0-128>
`status`	Whether to advertise this range; one of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Advertise this range
`PAL_FALSE`	Do not advertise this range

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PREFIXLEN_INVALID when the given prefix length is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_range_unset

This function removes the configured area range.

This function is called by the following command:

Syntax

```
int
ospf6_area_range_unset (u_int32_t vr_id, char *tag,
                        struct pal_in4_addr area_id,
                        void *addr, int prefixlen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`prefixlen`	Area range IPv6 prefix length <0-128>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PREFIXLEN_INVALID when the given prefix length is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_ERR_AREA_RANGE_NOT_EXIST when specified area range does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_stub_set

This function makes an area a normal area.

This function is called by the following command:

```
area stub
```

Syntax

```
int
ospf6_area_stub_set (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when specified area_id is a Backbone area

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_HAS_VLINK when the specified area has one or more vlinks

OSPF6_API_SET_ERR_AREA_IS_NSSA when the area is NSSA

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_area_stub_unset

This function unsets the area as stub configuration.

This function is called by the following command:

```
no area stub
```

Syntax

```
int
ospf6_area_stub_unset (u_int32_t vr_id, char *tag,
                       struct pal_in4_addr area_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when specified area_id is a Backbone area

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when specified area does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_auto_cost_reference_bandwidth_set

This function sets the reference bandwidth value. OSPF calculates the OSPF metric for an interface by dividing the reference bandwidth.

This function is called by the following command:

```
auto-cost reference-bandwidth <reference bandwith>
```

Syntax

```
int
```

```
ospf6_auto_cost_reference_bandwidth_set (u_int32_t vr_id,
                                         char *tag, int refbw);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`refbw`	The reference bandwidth in Mbits/second <1–4294967>.

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_REFERENCE_BANDWIDTH_INVALID when the bandwidth is not within the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_auto_cost_reference_bandwidth_unset

This function sets the reference bandwidth to its default value (100000 Kbps).

This function is called by the following command:

```
no auto-cost
```

Syntax

```
int
ospf6_auto_cost_reference_bandwidth_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`proc_id`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF63_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_capability_cspf_set

This function enables the CSPF capability for an OSPFv3 process.

This function is called by the following command:

```
capability cspf
```

Syntax

```
int
ospf6_capability_cspf_set(u_int_t vr_id, char* tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF6_API_SET_ERR_PROCESS_NOT_EXISTwhen the specified process does not exist

OSPF6_API_SET_ERR_CSPF_INSTANCE_EXIST when the specified CSPF instance does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_VR_NOT_EXISTwhen the virtual router cannot be found

OSPF6_API_SET_ERR_CSPF_CANT_START when CSPF cannot be started

ospf6_capability_cspf_unset

This function disables the CSPF capability for an OSPFv3 process.

This function is called by the following command:

```
no capability cspf
```

Syntax

```
int
ospf6_capability_cspf_unset(u_int_t vr_id, char* tag)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_PROCESS_NOT_EXISTwhen the specified process does not exist

OSPF6_API_SET_ERR_CSPF_INSTANCE_EXIST when the specified CSPF instance does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_VR_NOT_EXISTwhen the virtual router cannot be found

ospf6_default_metric_set

This function sets default metric values for the OSPFv3 routing protocol.

This function is called by the following command:

```
default-metric
```

Syntax

```
int
ospf6_default_metric_set (u_int32_t vr_id, char *tag, u_int32_t metric);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`metric`	Default metric <0-16777214>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_METRIC_INVALID when given metric is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_default_metric_unset

This function unsets the configured default metric.

This function is called by the following command:

```
no default-metric
```

Syntax

```
int
ospf6_default_metric_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_disable_db_summary_opt

This function disables the OSPFv3 Database Summary List optimization.

This function is called by the following command:

```
no enable db-summary-opt
```

Syntax

```
int
ospf6_disable_db_summary_opt (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_disable_db_summary_opt

This function disables the OSPFv3 Database Summary List optimization.

This function is called by the following command:

```
no enable db-summary-opt
```

Syntax

```
int
ospf6_disable_db_summary_opt (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_dna_set

This function is used to set DNA bit in LSA's Age before sending it out of the configured interface

This function is called by the following command:

```
ospfv3 flood-reduction
```

Syntax

```
int
ospf6_dna_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_dna_unset

This function is used to unset DNA bit in LSA's Age before sending it out of the configured interface

This function is called by the following command:

```
no ospfv3 flood-reduction
```

Syntax

```
int
ospf6_dna_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_enable_db_summary_opt

This function enables the OSPFv3 Database Summary List optimization.

This function is called by the following command:

```
enable db-summary-op
```

Syntax

```
int
ospf6_enable_db_summary_opt (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_cost_set

This function sets the current interface output cost. The configuration is stored regardless of whether or not a real interface exists.

This function is called by the following command:

```
ipv6 ospf cost
```

Syntax

```
int
ospf6_if_cost_set (u_int32_t vr_id, char *name, int instance_id,
                   u_int32_t cost);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`cost`	Interface output cost <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_COST_INVALID when the given cost is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_cost_unset

This function resets the configured output cost on the specified interface to the default value: 10.

This function is called by the following command:

```
no ipv6 ospf cost
```

Syntax

```
int
ospf6_if_cost_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_dc_set

This function enables demand circuit feature on the interface

This function is called by the following command:

```
ipv6 ospf demand-circuit
```

Syntax

```
int
ospf6_if_dc_set (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`instance_id`	The OSPFv3 instance id

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance ID does not exist

OSPF6_API_SET_ERR_INVALID_VALUE when the parameters do not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf6_if_dc_unset

This function disables demand-circuit feature on an interface.

This function is called by the following command:

```
no ipv6 ospf demand-circuit
```

Syntax

```
int
ospf6_if_dc_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name
`instance_id`	The OSPFv3 instance id

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance ID does not exist

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the parameters do not exist

OSPF_API_SET_SUCCESS when the function succeeds

ospf6_if_dead_interval_set

This function set the interval during which no hello packets are received and after which a neighbor is declared dead. Dead-interval is advertised in the Hello packets. When receiving Hello packets, OSPF router compares dead-interval in a receiving packet and the dead-interval configured on the receiving interface.

This function is called by the following command:

```
ipv6 ospf dead-interval
```

Syntax

```
int
ospf6_if_dead_interval_set (u_int32_t vr_id, char *name, int instance_id,
```

```
                                u_int32_t seconds);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`seconds`	Router dead interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_DEAD_INTERVAL_INVALID when given seconds is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_dead_interval_unset

This function resets the Router Dead Interval of the specified interface to the default value: 40 seconds.

This function is called by the following command:

```
no ipv6 ospf dead-interval
```

Syntax

```
int
ospf6_if_dead_interval_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_dna_set

This function sets the DNA bit in LSA's Age before sending it out of the configured interface.

This function is called by the following command:

```
ipv6 ospf flood-reduction
```

Syntax

```
int
ospf6_if_dna_set (u_int32_t vr_id, char *name, int instance_id, bool_t
fld_reduction_cfg);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name
`instance_id`	OSPFv3 instance Id
`fld_reduction_cfg`	Indicates if the flood-reduction is configured

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance ID does not exist

OSPF6_API_SET_ERR_INVALID_VALUE when the interface name does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_dna_unset

This function is used to unset DNA bit in LSA's Age before sending out of the configured interface

This function is called by the following command:

```
no ipv6 ospf flood-reduction
```

Syntax

```
int
ospf6_if_dna_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name
`instance_id`	OSPfv3 instance Id

Output Parameters

None

Return Values

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance ID does not exist

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the parameters do not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_hello_interval_set

This function configures Hello Interval on the specified interface.

This function is called by the following command:

```
ipv6 ospf hello-interval
```

Syntax

```
int
ospf6_if_hello_interval_set (u_int32_t vr_id, char *name, int instance_id,
                             u_int32_t seconds);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`seconds`	Hello interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when given seconds is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_hello_interval_unset

This function resets the configured Hello interval on a specified interface to the default value: 10 seconds.

This function is called by the following command:

```
no ipv6 ospf hello-interval
```

Syntax

```
int
ospf6_if_hello_interval_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_neighbor_set

This function configures the OSPFv3 routers interconnecting to non-broadcast networks.

This function is called by the following command:

```
ipv6 ospf neighbor
```

Syntax

```
int ospf6_if_neighbor_set (u_int32_t vr_id, char *name, int instance_id,
                           struct pal_in6_addr *addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`addr`	Neighbor address in an IPv6 format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_INVALID_VALUE when input parameter is invalid

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_neighbor_unset

This function removes the OSPFv3 routers configuration: interconnecting to non-broadcast networks.

This function is called by the following command:

```
no ipv6 ospf neighbor
```

Syntax

```
int ospf6_if_neighbor_unset (u_int32_t vr_id, char *name, int instance_id,
                             struct pal_in6_addr *addr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`addr`	neighbor address in an IPv6 format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_ERR_NBR_NOT_EXIST when the neighbor does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_network_set

This function sets the OSPF network type for the specified interface.

This function is called by the following command:

```
ipv6 ospf network
```

Syntax

```
int
ospf6_if_network_set (u_int32_t vr_id, char *name, int instance_id, int type);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`instance_id`	Interface instance ID <0–255>.
`type`	The network type; one of these constants from `lib/ospf_common/ospf_const.h`:

`OSPF_IFTYPE_POINTOPOINT`

`OSPF_IFTYPE_BROADCAST`

`OSPF_IFTYPE_NBMA`

`OSPF_IFTYPE_POINTOMULTIPOINT`

`OSPF_IFTYPE_POINTOMULTIPOINT_NBMA`

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_IF_ NETWORK_TYPE_INVALID when the network type is not supported

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_if_network_unset

This function removes the network type on the specified interface.

This function is called by the following command:

```
no ipv6 ospf network
```

Syntax

```
int
ospf_if_network_unset (u_int32_t vr_id, char *name int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	The interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the given interface is not configured

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_if_ipv6_router_set

This function enables OSPF routing on the specified interface.

This function is called by the following command:

```
ipv6 router ospf area
```

Syntax

```
int ospf6_if_ipv6_router_set (u_int32_t vr_id, char *name,
                              struct pal_in4_addr area_id, int format,
                              char *tag, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name
`area_id`	OSPFv3 area ID in an IPv4 address format
`tag`	OSPFv3 process tag
`format`	Area ID format: (1) Area ID format in IP Address (2) Area ID format as decimal

`instance_id`	Interface instance ID <0–255>

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_ID_FORMAT_INVALID when given format is outside the range

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_AREA when interface is already enabled for other area

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_PROCESS when interface is already enabled by other process

OSPF6_API_SET_ERR_IF_INSTANCE_ID_MISMTACH when interface is already enabled by other instance ID

OSPF6_API_SET_ERR_IF_NOT_EXIST when physical interface does not exist

OSPF6_API_SET_ERR_NO_LINKLOCAL_ADDRESS when physical interface does not have a linklocal IPv6 address

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_ipv6_router_unset

This function disables OSPF routing on a specified interface.

This function is called by the following command:

```
no ipv6 router ospf area
```

Syntax

```
int ospf6_if_ipv6_router_unset (u_int32_t vr_id, char *name,
                                struct pal_in4_addr area_id,
                                char *tag, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`area_id`	OSPFv3 area ID in an IPv4 address format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_AREA when interface is already enabled for other area

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_PROCESS when interface is already enabled by other process

OSPF6_API_SET_ERR_IF_NOT_EXIST when physical interface does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_link_lsa_suppression_set

This function enables or disables link LSA (type 8) suppression. A type 8 LSA gives information about link-local addresses and a list of IPv6 addresses on the link.

If enabled and the interface type is *not* broadcast or NBMA, the router does not send type 8 link LSAs. This implies that other routers on the link determine the router's next-hop address using a mechanism other than the type 8 link LSA. This feature is implicitly disabled if the interface type is broadcast or NBMA.

This function is called by the following command:

```
ipv6 ospf link-lsa-suppression
```

Syntax

```
int
ospf6_if_link_lsa_suppression_set (u_int32_t vr_id, char *name,
                                   int instance_id, int value)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name
`instance_id`	Interface instance ID <0–255>
`value`	Whether to enable link LSA suppression; one of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Enable link LSA suppression
`PAL_FALSE`	Disable link LSA suppression

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_priority_set

This function configures router priority on the specified interface.

This function is called by the following command:

```
ipv6 ospf priority
```

Syntax

```
int
```

```
ospf6_if_priority_set (u_int32_t vr_id, char *name, int instance_id,
                       u_int32_t priority);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`priority`	Router priority of the interface <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PRIORITY_INVALID when given priority is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_priority_unset

This function sets the router priority of the specified interface to its default value: 0.

This function is called by the following command:

```
no ipv6 ospf priority
```

Syntax

```
int
ospf6_if_priority_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_retransmit_interval_set

This function sets the interval between retransmission of Link State Update packets for adjacencies belonging to the interface.

This function is called by the following command:

```
ipv6 ospf retransmit-interval
```

Syntax

```
int
ospf6_if_retransmit_interval_set (u_int32_t vr_id, char *name,
                                  int instance_id, u_int32_t seconds);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`seconds`	Retransmit Interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when given seconds is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_retransmit_interval_unset

This function resets the interval between retransmission of Link State Update packets for adjacencies belonging to the interface to the default value: 5 seconds.

This function is called by the following command:

```
no ipv6 ospf retransmit-interval
```

Syntax

```
int
ospf6_if_retransmit_interval_unset (u_int32_t vr_id, char *name,
                                    int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_te_metric_set

This function sets the specified metric for OSPFv3 traffic engineering on an interface.

This function is called by the following command:

```
ipv6 te-metric
```

Syntax

```
int
ospf6_if_te_metric_set (u_int32_t vr_id, char *name, int instance_id,
                        u_int32_t metric);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`metric`	Interface TE metric <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_METRIC_INVALID when the given cost is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_te_metric_unset

This function unsets a TE metric for OSPFv3 traffic engineering on an interface.

This function is called by the following command:

```
no ipv6 te-metric
```

Syntax

```
int
ospf6_if_te_metric_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_transmit_delay_set

This function sets the estimated time it takes to transmit a Link State Update packet over the interface.

This function is called by the following command:

```
ipv6 ospf transmit-delay
```

Syntax

```
int
ospf6_if_transmit_delay_set (u_int32_t vr_id, char *name,
                             int instance_id, u_int32_t seconds);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.
`seconds`	Transmit Delay in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when given seconds is outside the range

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_if_transmit_delay_unset

This function resets the configured OSPF transmit delay of the specified interface to the default value: 1.

This function is called by the following command:

```
no ipv6 ospf transmit-delay
```

Syntax

```
int
ospf6_if_transmit_delay_unset (u_int32_t vr_id, char *name, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`name`	Interface name.
`instance_id`	Interface instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_ipv6_ospf_display_route_single_line_set

This function shows the results of `show ipv6 ospf route` in a single line.

This function is called by the following command:

```
ipv6 ospf display route single-line
```

Syntax

```
int
ospf6_ipv6_ospf_display_route_single_line_set (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_ipv6_ospf_display_route_single_line_unset

This function shows the results of `show ipv6 ospf route` in multiple lines.

This function is called by the following command:

```
no ipv6 ospf display route single-line
```

Syntax

```
int
ospf6_ipv6_ospf_display_route_single_line_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_max_concurrent_dd_set

This function sets the maximum number of concurrently processed database descriptors.

This function is called by the following command:

```
max-concurrent-dd <1-65535>
```

Syntax

```
int
ospf6_max_concurrent_dd_set (u_int32_t vr_id, char *tag, u_int16_t max_dd);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`max_dd`	The maximum of database descriptor processes <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_max_concurrent_dd_unset

This function sets the maximum concurrent database descriptors to its default value (5).

This function is called by the following command:

```
no ospf6_max_concurrent_dd_set
```

Syntax

```
int
ospf6_max_concurrent_dd_unset (u_int32_t vr_id, int proc_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

ospf6_passive_if_set

This function configures the specified interface into passive mode.

This function is called by the following command:

```
passive-interface
```

Syntax

```
int
ospf6_passive_if_set (u_int32_t vr_id, char *tag, char *name)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`name`	Interface name.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_passive_if_unset

This function unsets the passive interface configuration.

This function is called by the following command:

```
no passive-interface
```

Syntax

```
int
ospf6_passive_if_unset (u_int32_t vr_id, char *tag, char *name);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`name`	Interface name.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_metric_set

This function sets the metric for a redistributed route.

This function is called by the following commands:

```
redistribute metric
default-information originate metric
```

Syntax

```
int
ospf6_redistribute_metric_set (u_int32_t vr_id, char *tag,
                               int proto, void *stag, u_int32_t mvalue)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`proto`	Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_DEFAULT`
: System

`IPI_ROUTE_KERNEL`
: Kernel routes

`IPI_ROUTE_CONNECT`
: Connected routes

`IPI_ROUTE_STATIC`
: Static routes

`IPI_ROUTE_RIP`
: RIP routes

IPI_ROUTE_RIPNG
: RIPng routes

IPI_ROUTE_BGP
: BGP routes

IPI_ROUTE_ISIS
: IS-IS routes

IPI_ROUTE_OSPF
: OSPFv2 routes

IPI_ROUTE_OSPF6
: OSPFv3 routes

`stag` — OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

`mvalue` — Metric value for the external route <0-16777214>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is outside the range

OSPF6_API_SET_ERR_METRIC_INVALID when given metric is outside the range

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_metric_type_set

This function sets the metric type for a redistributed route.

This function is called by the following commands:

```
redistribute metric-type
default-information originate metric-type
```

Syntax

```
int
ospf6_redistribute_metric_type_set (u_int32_t vr_id, char *tag,
                                    int proto, void *stag, u_char mtype)
```

Input Parameters

`vr_id` — Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` — OSPFv3 process tag

`proto` — Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

IPI_ROUTE_DEFAULT

System

`IPI_ROUTE_KERNEL`

Kernel routes

`IPI_ROUTE_CONNECT`

Connected routes

`IPI_ROUTE_STATIC`

Static routes

`IPI_ROUTE_RIP`

RIP routes

`IPI_ROUTE_RIPNG`

RIPng routes

`IPI_ROUTE_BGP`

BGP routes

`IPI_ROUTE_ISIS`

IS-IS routes

`IPI_ROUTE_OSPF`

OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

`mtype` Metric type (see RFC 3101); one of these constants from `lib/ospf_common/ospf_const.h`:

`EXTERNAL_METRIC_TYPE_1`

OSPF External Type 1 metric.

`EXTERNAL_METRIC_TYPE_2`

OSPF External Type 2 metric.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_METRIC_TYPE_INVALID when given metric type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_metric_type_unset

This function sets the metric type of a redistributed route to its default value (2).

This function is called by the following commands:

```
no redistribute metric-type
no default-information originate metric-type
```

Syntax

```
int
ospf6_redistribute_metric_type_unset (u_int32_t vr_id, char *tag,
                                      int proto, void *stag)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_DEFAULT`
System

`IPI_ROUTE_KERNEL`
Kernel routes

`IPI_ROUTE_CONNECT`
Connected routes

`IPI_ROUTE_STATIC`
Static routes

`IPI_ROUTE_RIP`
RIP routes

`IPI_ROUTE_RIPNG`
RIPng routes

`IPI_ROUTE_BGP`
BGP routes

`IPI_ROUTE_ISIS`
IS-IS routes

`IPI_ROUTE_OSPF`
OSPFv2 routes

`IPI_ROUTE_OSPF6`
OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_metric_unset

This function sets the metric of a redistributed route to its default value (16777215).

This function is called by the following command:

```
no redistribute metric
no default-information originate metric
```

Syntax

```
int
ospf6_redistribute_metric_unset (u_int32_t vr_id, char *tag, int proto, void *stag)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_DEFAULT`
System

`IPI_ROUTE_KERNEL`
Kernel routes

`IPI_ROUTE_CONNECT`
Connected routes

`IPI_ROUTE_STATIC`
Static routes

`IPI_ROUTE_RIP`
RIP routes

`IPI_ROUTE_RIPNG`
RIPng routes

`IPI_ROUTE_BGP`
BGP routes

`IPI_ROUTE_ISIS`
IS-IS routes

`IPI_ROUTE_OSPF`
OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_set

This function redistributes routes from the specified protocol.

This function is called by the following commands:

```
passive-interface
redistribute
```

Syntax

```
int
ospf6_redistribute_set (u_int32_t vr_id, char *tag, int proto, void *stag)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_KERNEL`
Kernel routes

`IPI_ROUTE_CONNECT`
Connected routes

`IPI_ROUTE_STATIC`
Static routes

`IPI_ROUTE_RIP`
RIP routes

`IPI_ROUTE_RIPNG`
RIPng routes

`IPI_ROUTE_BGP`
BGP routes

`IPI_ROUTE_ISIS`

IS-IS routes

`IPI_ROUTE_OSPF`

OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_SELF_TAG when trying to redistribute its own routes into its own routing database (self redistribution)

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_redistribute_unset

This function stops redistributing routes from the specified protocol.

This function is called by the following command:

```
no redistribute
```

Syntax

```
int
ospf6_redistribute_unset (u_int32_t vr_id, char *tag, int proto, void *stag
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_KERNEL`

Kernel routes

`IPI_ROUTE_CONNECT`

Connected routes

`IPI_ROUTE_STATIC`

Static routes

`IPI_ROUTE_RIP`

RIP routes

`IPI_ROUTE_RIPNG`

RIPng routes

`IPI_ROUTE_BGP`

BGP routes

`IPI_ROUTE_ISIS`

IS-IS routes

`IPI_ROUTE_OSPF`

OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_routemap_set

This function assigns a route-map to a redistributed protocol.

This function is called by the following command:

```
redistribute route-map
```

Syntax

```
int
ospf6_routemap_set (u_int32_t vr_id, char *tag, int proto,
                    void *stag, char *name)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_KERNEL`

Kernel routes

`IPI_ROUTE_CONNECT`

Connected routes

`IPI_ROUTE_STATIC`

Static routes

`IPI_ROUTE_RIP`

RIP routes

`IPI_ROUTE_RIPNG`

RIPng routes

`IPI_ROUTE_BGP`

BGP routes

`IPI_ROUTE_ISIS`

IS-IS routes

`IPI_ROUTE_OSPF`

OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

`name` Route-map name

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_routemap_unset

This function removes a route map from a redistributed protocol.

This function is called by the following command:

```
no redistribute route-map
```

Syntax

```
int
ospf6_routemap_unset (u_int32_t vr_id, char *tag, int proto, void *stag))
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`tag` OSPFv3 process tag

`proto` Redistributed protocol type; one of the constants from `pal/api/pal_types.def`:

`IPI_ROUTE_KERNEL`

Kernel routes

`IPI_ROUTE_CONNECT`

Connected routes

`IPI_ROUTE_STATIC`

Static routes

`IPI_ROUTE_RIP`

RIP routes

`IPI_ROUTE_RIPNG`

RIPng routes

`IPI_ROUTE_BGP`

BGP routes

`IPI_ROUTE_ISIS`

IS-IS routes

`IPI_ROUTE_OSPF`

OSPFv2 routes

`IPI_ROUTE_OSPF6`

OSPFv3 routes

`stag` — OSPFv3 process tag or OSPF process identifier used to redistribute from one OSPF instance to another OSPF instance

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_INVALID when the given protocol type is not valid

OSPF6_API_SET_ERR_REDISTRIBUTE_PROTO_TAG_INVALID when the given process tag does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_router_id_set

This function configures the OSPFv3 Router ID.

This function is called by the following command:

```
router-id
```

Syntax

```
int
ospf6_router_id_set (u_int32_t vr_id, char *tag,
                     struct pal_in4_addr router_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`router_id`	Router ID in IPv4 address format.<A.B.C.D>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_INVALID_ROUTER_ID when given router ID is unspecified (equals 0.0.0.0)

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_router_id_unset

This function forces OSPFv3 to stop the routing functionality.

This function is called by the following command:

```
no router-id
```

Syntax

```
int
ospf6_router_id_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_router_set

This function initiates the OSPFv3 router instance.

This function is called by the following command:

```
router ipv6 ospf
```

Syntax

```
int
```

```
ospf6_router_set (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_CANT_INITIATE when the process cannot be initiated properly

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_router_unset

This function removes the OSPFv3 router instance.

This function is called by the following command:

```
no router ipv6 ospf
```

Syntax

```
int
ospf6_router_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_frr_set

This function enables or disables fast rerouting on all OSPFv3 interfaces.

This function is called by the following command:

```
(no)fast-reroute keep-all-paths
```

Syntax

```
int
ospf6_frr_set (u_int32_t vr_id, char* tag, bool_t frr_set);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`tag`	OSPFv3 process tag
`frr_set`	One of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Enable fast rerouting.
`PAL_FALSE`	Disable fast rerouting.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when `tag` cannot be found

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_frr_interface_set

This function permits the interface to be used as the next hop in a repair path or prohibits the interface from being used as the next hop in a repair path.

This function is called by the following command:

```
(no) ipv6 ospf fast-reroute per-prefix candidate disable (instance-id <0-255>)
```

Syntax

```
int
ospf6_frr_interface_set (u_int32_t vr_id, char* name, struct interface *ifp,
                         int instance_id, bool_t frr_if_set)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`name`	Name of the interface
`ifp`	Pointer to the interface.
`instance_id`	Instance ID of the interface
`frr_if_set`	One of these constants from `pal/linux/pal_types.h`:
`PAL_TRUE`	Enable fast rerouting for the interface.
`PAL_FALSE`	Disable fast rerouting for the interface

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance ID is not in the specified range

OSPF6_API_SET_ERR_INVALID_VALUE when `tag` cannot be found

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_frr_tie_break_prority_set

This function sets and unsets the priority for a type of fast reroute repair path.

This function is called by the following command:

```
(no) fast-reroute tie-break <protection-type> index <1-4>
```

The default priorities are:

1. PRIMARY_PATH
2. INTERFACE_DISOINT
3. NODE_PROTECTING
4. BROADCAST_INTERFACE_DISJOINT

Syntax

```
ospf6_frr_tie_break_prority_set(u_int32_t vr_id, char* tag, enum ospf6_frr_pref val,
                                u_int32_t index, bool_t isSet)
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0.

`tag` OSFv3 process tag

`val` The type of repair path; one of these constants from `ospf6d/ospf6d.h`:

`PRIMARY_PATH`

When there are multiple loop-free alternate paths. The primary path is selected for an frr if it is node-protecting as well as link-protecting

`NODE_PROTECTING`

Bypass the `primary-path` gateway router which might not protect the router that is the next hop in the primary path.

`INTERFACE_DISJOINT`

Do not select point-to-point interfaces that have no alternate next hop for rerouting if the primary gateway fails, thus protecting the interface.

`BROADCAST_INTERFACE_DISJOINT`

Do not use the interface if connected to a broadcast network. Repair paths protect links when a repair path and a protected primary path use *different* next-hop interfaces. However, on broadcast interfaces, if the repair path is computed via the same interface as the primary path, but their next-hop gateways are different, the router is protected but the link might not be.

`index` Tiebreak priority `<1-4>`. A lower value has higher preference.

`isSet` Whether to set the priority for the repair path type specified by `val` to its default value or to a different value; one of these constants from `ospf6d/ospf6_api.h`:

`OSPF6_FRR_TIE_BREAK_SET`

Set the repair path priority to the value specified by `index`

`OSPF6_FRR_TIE_BREAK_UNSET`

Set the repair path priority to its default value

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when `tag` cannot be found

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_frr_tie_break_priority_all_unset

The function unsets the user configured tie-breaking priority and resets it to its default priorities.

This function is called by the following command:

```
no fast-reroute tie-break
```

Syntax

```
int32_t
ospf6_frr_tie_break_prority_all_unset(u_int32_t vr_id, char* tag)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0.
`tag`	OSPFv3 process tag

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_not_advertise_set

This function suppresses external routes that match a specified address range.

This function is called by the following commands:

```
summary-address X:X::X:X/M not-advertise
summary-address A.B.C.D/M not-advertise
```

Syntax

```
int
ospf6_summary_address_not_advertise_set (u_int32_t vr_id, char *tag,
                                         void *addr, u_char masklen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_not_advertise_unset

This function stops suppressing external routes that match a specified address range.

This function is called by the following commands:

```
no summary-address X:X::X:X/M not-advertise
no summary-address A.B.C.D/M not-advertise
```

Syntax

```
int
ospf6_summary_address_not_advertise_unset (u_int32_t vr_id, char *tag,
                                           void *addr,
                                           u_char masklen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_set

This function summarizes external routes with the specified address range.

An address range is a pairing of a starting address and a mask that is almost the same as IP network number. For example:

- If the specified IPV6 address range is 2020:100:100:2000::/53, it matches 2020:100:100:2222::/64, 2020:100:100:2666::/64 and so on.
- If the specified IPV4 address range is 192.168.0.0/255.255.240.0, it matches 192.168.1.0/24, 192.168.4.0/22, 192.168.8.128/25 and so on.

Redistributing routes from other protocols into OSPF requires the router to advertise each route individually in an external LSA. Use this function to advertise one summary route for all redistributed routes covered by a specified network address and mask. This minimizes the size of the OSPF link state database.

This function is called by the following commands:

```
summary-address X:X::X:X/M
summary-address A.B.C.D/M
```

Syntax

```
int
ospf6_summary_address_set (u_int32_t vr_id, char *tag,
                           void *addr, u_char masklen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_EXIST when a summary address is already set

OSPF6_API_SET_MALLOC_ERR when there was a memory allocation error

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_tag_set

This function sets a tag value to use as a “match” value for controlling redistribution via route maps.

This function is called by the following commands:

```
summary-address A.B.C.D/M tag <0-4294967295>
summary-address X:X::X:X/M tag <0-4294967295>
```

Syntax

```
int
ospf6_summary_address_tag_set (u_int32_t vr_id, char *tag,
                               void *addr, u_char masklen,
                               u_int32_t route_tag)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length
`route_tag`	Route tag value

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_tag_unset

This function removes a tag value to use as a “match” value for controlling redistribution via route maps.

This function is called by the following commands:

```
no summary-address A.B.C.D/M tag <0-4294967295>
no summary-address X:X::X:X/M tag <0-4294967295>
```

Syntax

```
int
ospf6_summary_address_tag_unset (u_int32_t vr_id, char *tag,
                                 void * addr, u_char masklen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_summary_address_unset

This function removes a summary address.

This function is called by the following commands:

```
no summary-address X:X::X:X/M
no summary-address A.B.C.D/M
```

Syntax

```
int
ospf6_summary_address_unset (u_int32_t vr_id, char *tag,
                             void *addr, u_char masklen)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`addr`	IPv6 or IPv4 starting address; this parameter is defined as a void pointer so you can cast it to either the `prefix_ipv4` or `prefix_ipv6` structure in `lib/prefix.h`
`masklen`	Mask length

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_timers_spf_set

This function sets the minimum and maximum delay between a topology change, being either received in an LSA or self detected and the SPF being run.

This function is called by the following command:

```
timers spf exp
```

Syntax

```
int
ospf6_timers_spf_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; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`min_delay`	The minimum SPF hold delay time in milliseconds.
`max_delay`	The maximum SPF hold delay time in milliseconds.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_timers_spf_unset

This function resets the SPF minimum and maximum delay to their default values:

- Minimum default delay: 500 milliseconds
- Maximum default delay: 50000 milliseconds

This function is called by the following command:

```
no timers spf exp
```

Syntax

```
int
ospf6_timers_spf_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_dead_interval_set

This function configures the Router Dead Interval of the specified virtual link.

This function is called by the following command:

```
area virtual-link dead-interval
```

Syntax

```
int
ospf6_vlink_dead_interval_set (u_int32_t vr_id, char *tag,
                               struct pal_in4_addr area_id,
                               struct pal_in4_addr peer_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`interval`	Router dead interval <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_DEAD_INTERVAL_INVALID when given interval is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_dead_interval_unset

This function unsets the configured Router Dead Interval of the specified virtual link.

This function is called by the following command:

```
no area virtual-link dead-interval
```

Syntax

```
int
ospf6_vlink_dead_interval_unset (u_int32_t vr_id, char *tag,
                                 struct pal_in4_addr area_id,
                                 struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_format_set

This function specifies the format for the specified virtual link configuration.

This function is called by the following command:

```
area ID
```

Syntax

```
int
ospf6_vlink_format_set (u_int32_t vr_id, char *tag,
                        struct pal_in4_addr area_id,
                        struct pal_in4_addr peer_id, int format);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`format`	Area ID format: (1) Area ID format in IP Address (2) Area ID format as decimal

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_ID_FORMAT_INVALID when given area ID format is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_hello_interval_set

This function configures Hello Interval on the specified virtual link.

This function is called by the following command:

```
area virtual-link hello-interval
```

Syntax

```
int
ospf6_vlink_hello_interval_set (u_int32_t vr_id,
                                char *tag, struct pal_in4_addr area_id,
                                struct pal_in4_addr peer_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`interval`	The interval (seconds) before the router sends the hello packet <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when given interval is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_hello_interval_unset

This function resets the Hello Interval on a specified virtual link to the default value: 10 seconds.

This function is called by the following command:

```
no area virtual-link hello-interval
```

Syntax

```
int
ospf6_vlink_hello_interval_unset (u_int32_t vr_id,
                                  char *tag, struct pal_in4_addr area_id,
                                  struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_instance_id_set

This function configures the instance ID on the specified virtual link.

This function is called by the following command:

```
area virtual-link instance-id
```

Syntax

```
int
ospf6_vlink_instance_id_set (u_int32_t vr_id, char *tag,
                             struct pal_in4_addr area_id,
                             struct pal_in4_addr peer_id, int instance_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`instance_id`	Instance ID <0–255>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_IF_INSTANCE_ID_INVALID when the given instance ID is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_instance_id_unset

This function resets the configured instance ID of the specified virtual link to the default value: 0.

This function is called by the following command:

```
no area virtual-link instance-id
```

Syntax

```
int
```

```
ospf6_vlink_instance_id_unset (u_int32_t vr_id, char *tag,
                               struct pal_in4_addr area_id,
                               struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_retransmit_interval_set

This function sets the retransmit interval on the specified virtual link. The router waits before it retransmits the packet.

This function is called by the following command:

```
area virtual-link retransmit-interval
```

Syntax

```
int
ospf6_vlink_retransmit_interval_set (u_int32_t vr_id, char *tag,
                                     struct pal_in4_addr area_id,
                                     struct pal_in4_addr peer_id,
                                     int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`interval`	Retransmit Interval in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when given interval is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_retransmit_interval_unset

This function resets the configured retransmit interval of the specified virtual link to the default value: 5.

This function is called by the following command:

```
no area virtual-link retransmit-interval
```

Syntax

```
int
ospf6_vlink_retransmit_interval_unset (u_int32_t vr_id, char *tag,
                                       struct pal_in4_addr area_id,
                                       struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_set

This function configures the virtual link as specified.

This function is called by the following command:

```
area vlink
```

Syntax

```
int
ospf6_vlink_set (u_int32_t vr_id, char *tag,
                 struct pal_in4_addr area_id, struct pal_in4_addr peer_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_AREA_NOT_DEFAULT when specified area is stub or NSSA

OSPF6_API_SET_ERR_VLINK_CANT_GET when virtual link is not initiated

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_transmit_delay_set

This function configures the transmit delay on a specified virtual link.

This function is called by the following command:

```
area virtual-link transmit-delay
```

Syntax

```
int
ospf6_vlink_transmit_delay_set (u_int32_t vr_id, char *tag,
                                struct pal_in4_addr area_id,
                                struct pal_in4_addr peer_id, int interval);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`area_id`	OSPFv3 area ID in an IPv4 address format.
`peer_id`	Neighbor ID.
`interval`	Transmit Delay in seconds <1–65535>.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when given interval is outside the range

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_transmit_delay_unset

This function unsets the configured Transmit Delay on the specified virtual link.

This function is called by the following command:

```
no area virtual-link transmit-delay
```

Syntax

```
int
ospf6_vlink_transmit_delay_unset (u_int32_t vr_id, char *tag,
                                  struct pal_in4_addr area_id,
                                  struct pal_in4_addr peer_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
tag	OSPFv3 process tag.
area_id	OSPFv3 area ID in an IPv4 address format.
peer_id	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_vlink_unset

This function unsets the virtual link configuration.

This function is called by the following command:

```
no area vlink
```

Syntax

```
int
ospf6_vlink_unset (u_int32_t vr_id, char *tag, struct pal_in4_addr area_id,
                   struct pal_in4_addr peer_id);
```

Input Parameters

vr_id	Virtual router ID; for a non-virtual-router implementation, specify 0
tag	OSPFv3 process tag.
area_id	OSPFv3 area ID in an IPv4 address format.
peer_id	Neighbor ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_VLINK_NOT_EXIST when the specified virtual link does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPFv3 Graceful Restart API

Following is a summary of the OSPFv3 graceful restart API functions. Details are provided in the following subsections. For more information about the OSPF graceful restart feature, refer to Chapter 2, *Graceful Restart*.

Function	Description
ospf6_capability_restart_set	Enables the OSPFv3 graceful restart capability
ospf6_capability_restart_unset	Disables the OSPFv3 graceful restart capability
ospf6_graceful_restart_set	Configures the grace period (in seconds) for restarting the router
ospf6_graceful_restart_unset	Resets the grace period to the default value
ospf6_restart_helper_grace_period_set	Sets the configured maximum grace period for a neighbor to act as helper
ospf6_restart_helper_grace_period_unset	Reverts to the default value
ospf6_restart_helper_never_router_id_set	Adds the specified router ID to the Never-Router ID list
ospf6_restart_helper_never_router_id_unset	Deletes the specified router ID from the Never-Router ID list
ospf6_restart_helper_never_router_unset_all	Deletes the entire Never-Router ID list
ospf6_restart_helper_policy_set	Configures the specified helper policy
ospf6_restart_helper_policy_unset	Resets helper to “always accept” (default)
ospf6_restart_helper_policy_unset_all	Reverts to the default setting

Include File

To call the functions in this section, include `ospf6d\ospf6_api.h`.

ospf6_capability_restart_set

This function enables the OSPFv3 graceful restart capability.

This function is called by the following command:

```
capability restart graceful
```

Syntax

```
int
ospf6_capability_restart_set (u_int32_t vr_id, char *tag, int method);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.
`method`	Graceful restart. By default, a router is graceful-restart capable.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

ospf6_capability_restart_unset

This function disables the OSPFv3 graceful restart capability.

This function is called by the following command:

```
no capability restart
```

Syntax

```
int
ospf6_capability_restart_unset (u_int32_t vr_id, char *tag);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`tag`	OSPFv3 process tag.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

ospf6_graceful_restart_set

This function sets the grace period (in seconds) for restarting the router.

This function is called by the following command:

```
ipv6 ospf restart grace-period
```

Syntax

```
int
ospf6_graceful_restart_set (u_int32_t vr_id, int seconds, int reason);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`seconds`	Grace period <1–1800>.
`reason`	Reason for restart; one of these constants from `ospf6d/ospf6d.h`:

`OSPF6_RESTART_REASON_UNKNOWN`

`OSPF6_RESTART_REASON_SOFTWARE`

`OSPF6_RESTART_REASON_UPGRADE`

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_GRACE_PERIOD_INVALID when an invalid grace period is set

OSPF6_API_SET_ERR_INVALID_REASON if reason for restart is unknown

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_graceful_restart_unset

This function resets the grace period to the default value: 120 seconds.

This function is called by the following command:

```
no ipv6 ospf restart grace-period
```

Syntax

```
int
ospf6_graceful_restart_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_graceful

This function causes a graceful restart of the router. After this command executes, the router immediately shuts down and notifies NSM that the shut down was graceful. In turn, NSM preserves routes installed by OSPFv3 until the grace period expires.

This function is called by the following command:

```
restart ipv6 ospf graceful
```

Syntax

```
int
ospf6_restart_graceful (u_int32_t vr_id, u_int16_t grace_period,
                        int reason)
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`grace_period`	Grace period in seconds <1-1800>
`reason`	Reason for restart; one of these constants from `ospf6d/ospf6d.h`:
	`OSPF6_RESTART_REASON_UNKNOWN`
	`OSPF6_RESTART_REASON_SOFTWARE`
	`OSPF6_RESTART_REASON_UPGRADE`

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_GRACE_PERIOD_INVALID when the grace period is out of range

OSPF6_API_SET_ERR_INVALID_REASON when the reason is invalid

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_grace_period_set

This function sets the configured maximum grace period for a neighbor to act as helper.

This function is called by the following command:

```
ipv6 ospf restart helper max-grace-period
```

Syntax

```
int
ospf6_restart_helper_grace_period_set (u_int32_t vr_id, int seconds);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`seconds`	Grace period <1–1800>

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_GRACE_PERIOD_INVALID when an invalid grace period is set

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_grace_period_unset

This function reverts the configured maximum grace period for a neighbor to act as helper to the default value: 1.

This function is called by the following command:

```
no ipv6 ospf restart helper max-grace-period
```

Syntax

```
int
ospf6_restart_helper_grace_period_unset (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_never_router_id_set

This function adds the specified router ID to the Never-Router ID list.

This function is called by the following command:

```
ipv6 ospf restart helper never router-id
```

Syntax

```
int
ospf6_restart_helper_never_router_id_set (u_int32_t vr_id,
struct pal_in4_addr router_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`router_ID`	The router ID in an IPv4 format.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_ROUTER_ID_ALREADY_CONFIGURED when the specified router ID is already configured

OSPF6_API_SET_ERR_MALLOC_FAIL_FOR_ROUTERID if the memory allocation fails for the new router ID

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_never_router_id_unset

This function deletes the specified router ID from the Never-Router ID list.

This function is called by the following command:

```
no ipv6 ospf restart helper never
```

Syntax

```
int
ospf6_restart_helper_never_router_id_unset (u_int32_t vr_id,
struct pal_in4_addr router_id_ptr);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0
`router_id_ptr`	Pointer to the router ID.

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_EMPTY_NEVER_RTR_ID when the Never-Router ID list is empty

OSPF6_API_SET_ERR_NEVER_RTR_ID_NOT_EXIST when the specified router ID does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_never_router_unset_all

This function deletes the entire Never-Router ID list.

This function is called by the following command:

```
no ipv6 ospf restart helper all
```

Syntax

```
int
ospf6_restart_helper_never_router_unset_all (u_int32_t vr_id);
```

Input Parameters

`vr_id`	Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_EMPTY_NEVER_RTR_ID when the Never-Router ID list is empty

OSPF6_API_SET_SUCCESS when the function succeeds

ospf6_restart_helper_policy_set

This function configures the specified helper policy.

This function is called by the following command:

```
ipv6 ospf restart helper
```

Syntax

```
int
ospf6_restart_helper_policy_set (u_int32_t vr_id, int policy);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`policy` Helper policy; one of these constants from `ospf6d/ospf6d.h`:

`OSPF6_RESTART_HELPER_NEVER`
Never act as helper (overwrite the existing policy)

`OSPF6_RESTART_HELPER_ONLY_RELOAD`
Help only on software reloads

`OSPF6_RESTART_HELPER_ONLY_UPGRADE`
Help only on software upgrades

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_INVALID_HELPER_POLICY when the specified helper policy is set

ospf6_restart_helper_policy_unset

This function resets the configuration to the default setting of `Never` helper policy.

This function is called by the following command:

```
no ipv6 ospf restart helper
```

Syntax

```
int
ospf6_restart_helper_policy_unset (u_int32_t vr_id, int policy);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

`policy` Helper policy; one of these constants from `ospf6d/ospf6d.h`:

`OSPF6_RESTART_HELPER_NEVER`
Never act as helper (overwrite the existing policy)

`OSPF6_RESTART_HELPER_ONLY_RELOAD`
Help only on software reloads

`OSPF6_RESTART_HELPER_ONLY_UPGRADE`
Help only on software upgrades

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_INVALID_HELPER_POLICY when an invalid helper policy is set

ospf6_restart_helper_policy_unset_all

This function resets all configured helper policies.

This function is called by the following command:

```
no ipv6 ospf restart helper all
```

Syntax

```
int
ospf6_restart_helper_policy_unset_all (u_int32_t vr_id);
```

Input Parameters

`vr_id` Virtual router ID; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_SUCCESS when the function succeeds

CHAPTER 14 OSPFv2 SNMP API

This chapter describes the SNMP functions for the OSPFv2 management information base (MIB).

Overview of MIB Implementation

The `OSPF-MIB.txt` file contains the MIB definitions for SNMP (based on the definitions in RFC 1850). This file is divided into these sections:

- General variables – parameters that apply globally to the Router's OSPF Process
- OSPF Area Table – describes the OSPF areas in which the routers participate
- OSPF Area Default Metric Table – describes the metrics that a default Area Border Router will advertise into a Stub area
- OSPF Link State Database—contains the Link State Advertisements to which the device is attached
- Address Range Table – describes the Address Range Summaries that are configured to be propagated from an area to reduce the area's exposure beyond its borders
- OSPF Host Table – indicates which hosts are directly attached to the Router, and which metrics and types of
- service should be advertised
- OSPF Interface Table – augments the ipAddrTable with OSPF-specific information
- OSPF Interface Metric Table – the metrics to be advertised for a specified interface at the various types of service
- OSPF Virtual Interface Table – describes the virtual links for which the OSPF Process is configured
- OSPF Neighbor Table – describes all neighbors to the OSPF process
- OSPF Virtual Neighbor Table – describes all virtual neighbors to the OSPF process
- External Link State Database – contains the Link State Advertisements from all the areas to which the device is attached
- OSPF Area Aggregate Table – describes prefixes which summarize routing information for export outside an area conformance data

The `OSPF-TRAP-MIB.txt` file contains the MIB definitions for SNMP (based on the definitions in RFC 1850). This file is divided into these sections:

- trap support objects — describes the OSPF trap support objects
- traps — describes the traps
- conformance data

The `OSPF-PRIVATE-MIB.txt` file contains the MIB definitions related to NSSA defined in the OSPF area table in RFC 4750. As described in NSSA Support, ZebOS-XP supports only the OSPF area table from RFC 4750 using this private MIB.

General Variables

ospfBasicGroup

Attribute	Syntax	Access	Function
ospfRouterId	RouterID	read-write	ospf_get_router_id
ospfExtLsdbLimit	Integer32	read-write	ospf_get_ext_lsdb_limit

Functions

Following is a summary of the OSPFv2 functions. Details are provided in the following subsections.

Function	Description
ospf_get_router_id	Gets the unique 32-bit ID for the router.
ospf_get_ext_lsdb_limit	Gets the number that represents the maximum number of non-default AS-external-LSA entries that can be stored in the link-state database

ospf_get_router_id

This function gets the unique 32-bit ID for the router.

Syntax

```
int
ospf_get_router_id (int proc_id, struct pal_in4_addr *ret, u_int32_t vr_id);
```

Input Parameters

`proc_id`	An integer that contains the OSPF process ID
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	The router ID

Return Value

OSPF_API_GET_SUCCESS

ospf_get_ext_lsdb_limit

This function gets the number that represents the maximum number of non-default AS-external-LSA entries that can be stored in the link-state database. When the HAVE_OSPF_DB_OVERFLOW compile time option is enabled, this call gets the actual value; when not enabled, it returns -1 for no stated limit.

Syntax

```
int
ospf_get_ext_lsdb_limit (int proc_id, int *ret, u_int32_t vr_id);
```

Input Parameters

`proc_id`	An integer that contains the OSPF process ID
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	The LIMIT to the number of non-default AS-external-LSA entries

Return Value

OSPF_API_GET_SUCCESS

NSSA Support

ZebOS-XP provides partial support of RFC 4750 through a private MIB defined in `OSPF-PRIVATE-MIB.txt` with this object identifier:

- 1.3.6.1.4.1.36673.1.2.1 (Internet.Private.Enterprise.IpInfusion.mgmt.L3.OSPF)

This MIB defines the complete ospfAreaEntry group of RFC4750.

Area Entry Group

The ospfAreaEntry group contains NSSA translation parameters and statistics for a router's attached area.

Attribute	Syntax	Access	Function
ospfAreaNssaTranslatorRole	INTEGER	read-create	ospf_get_area_nssa_translator_role ospf_get_next_area_nssa_translator_role ospf_set_nssa_translator_role
ospfAreaNssaTranslatorState	INTEGER	read-only	ospf_get_area_nssa_translator_state ospf_get_next_area_nssa_translator_state
ospfAreaNssaTranslatorStability Interval	PositiveInteger	read-create	ospf_get_area_nssa_translator_stability_interval ospf_get_next_area_nssa_translator_stability_interval ospf_set_nssa_stability_interval
ospfAreaNssaTranslatorEvents	Counter32	read-only	ospf_get_area_nssa_translator_events ospf_get_next_area_nssa_translator_events

Functions

Following is a summary of the NSSA MIB functions. Details are provided in the following subsections.

Function	Description
ospf_get_area_nssa_translator_events	Gets the number of translator state changes that have occurred since the last boot-up
ospf_get_area_nssa_translator_role	Determines the router's ability to translate type-7 LSAs into type-5 LSAs
ospf_get_area_nssa_translator_stability_interval	Gets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required
ospf_get_area_nssa_translator_state	Determines whether and how an NSSA border router is translating type-7 LSAs into type-5 LSAs
ospf_get_next_area_nssa_translator_events	Gets the number of translator state changes that have occurred since the last boot-up
ospf_get_next_area_nssa_translator_role	Determines the router's ability to translate type-7 LSAs into type-5 LSAs
ospf_get_next_area_nssa_translator_stability_interval	Gets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required
ospf_get_next_area_nssa_translator_state	Determines whether and how an NSSA border router is translating type-7 LSAs into type-5 LSAs
ospf_set_nssa_translator_role	Sets the router's ability to translate type-7 LSAs into type-5 LSAs
ospf_set_nssa_stability_interval	Sets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required

ospf_get_area_nssa_translator_events

This function gets the number of translator state changes that have occurred since the last boot-up.

Syntax

```
int
ospf_get_area_nssa_translator_events (int proc_id, struct pal_in4_addr area_id,
                                      int *ret, u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	Translator state changes

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_area_nssa_translator_role

This function determines the router's ability to translate type-7 LSAs into type-5 LSAs.

Syntax

```
int
ospf_get_area_nssa_translator_role (int proc_id, struct pal_in4_addr area_id,
                                    int *ret, u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret` Translator role; one of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATE_CANDIDATE`
Translate type-7 LSAs into type-5 LSAs if router is elected

`OSPF_NSSA_TRANSLATE_ALWAYS`
Always translate type-7 LSAs into type-5 LSAs LSA

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_area_nssa_translator_stability_interval

This function gets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required.

Syntax

```
int
ospf_get_area_nssa_translator_stability_interval (int proc_id,
                                                  struct pal_in4_addr area_id,
                                                  int *ret, u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	Translator stability interval in seconds <0-2147483647>

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_area_nssa_translator_state

This function determines whether and how an NSSA border router is translating type-7 LSAs into type-5 LSAs.

Syntax

```
int
ospf_get_area_nssa_translator_state (int proc_id, struct pal_in4_addr area_id,
                                     int *ret, u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	One of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATOR_DISABLED`

`OSPF_NSSA_TRANSLATOR_ENABLED`

`OSPF_NSSA_TRANSLATOR_ELECTED`

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_next_area_nssa_translator_events

This function gets the number of translator state changes that have occurred since the last boot-up.

Syntax

```
int
ospf_get_next_area_nssa_translator_events (int proc_id,
                                           struct pal_in4_addr *area_id,
                                           int indexlen, int *ret,
                                           u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Index length
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	Translator state changes

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_next_area_nssa_translator_role

This function determines the router's ability to translate type-7 LSAs into type-5 LSAs.

Syntax

```
int
ospf_get_next_area_nssa_translator_role (int proc_id,
                                         struct pal_in4_addr *area_id,
                                         int indexlen, int *ret,
                                         u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Index length
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret` Translator role; one of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATE_CANDIDATE`
Translate type-7 LSAs into type-5 LSAs if router is elected

`OSPF_NSSA_TRANSLATE_ALWAYS`

Always translate type-7 LSAs into type-5 LSAs LSA

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_next_area_nssa_translator_stability_interval

This function gets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required.

Syntax

```
int
ospf_get_next_area_nssa_translator_stability_interval (int proc_id,
                                                       struct pal_in4_addr *area_id,
                                                       int indexlen, int *ret,
                                                       u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Index length
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

`ret`	Translator stability interval in seconds <0-2147483647>

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_get_next_area_nssa_translator_state

This function determines whether and how an NSSA border router is translating type-7 LSAs into type-5 LSAs.

Syntax

```
int
ospf_get_next_area_nssa_translator_state (int proc_id,
                                          struct pal_in4_addr *area_id,
                                          int indexlen, int *ret,
                                          u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Index length

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

Output Parameters

`ret` One of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATOR_DISABLED`

`OSPF_NSSA_TRANSLATOR_ENABLED`

`OSPF_NSSA_TRANSLATOR_ELECTED`

Return Values

OSPF_API_GET_SUCCESS when the function succeeds

OSPF_API_GET_ERROR when the function fails

ospf_set_nssa_translator_role

This function sets the router's ability to translate type-7 LSAs into type-5 LSAs.

Syntax

```
int
ospf_set_nssa_translator_role (int proc_id, struct pal_in4_addr area_id, int val,
                      u_int32_t vr_id)
```

Input Parameters

`proc_id` OSPF process identifier

`area_id` Area identifier in an IPv4 address format

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

`val` Translator role; one of these constants from `ospfd/ospfd.h`:

`OSPF_NSSA_TRANSLATE_CANDIDATE`
Translate type-7 LSAs into type-5 LSAs if router is elected

`OSPF_NSSA_TRANSLATE_ALWAYS`
Always translate type-7 LSAs into type-5 LSAs LSA

Output Parameters

None

Return Values

OSPF_API_SET_ERR_WRONG_VALUE when `val` is not valid

OSPF_API_SET_ERR_AREA_NOT_EXIST when the given area cannot be found

OSPF_API_SET_ERR_PROCESS_ID_INVALID when the process ID is not valid

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not a not-so-stubby-area

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_SUCCESS when the function succeeds

ospf_set_nssa_stability_interval

This function sets the number of seconds an elected translator should continue to perform its translation duties after it has determined its services are no longer required.

Syntax

```
int
ospf_set_nssa_stability_interval (int proc_id, struct pal_in4_addr area_id, int val,
                        u_int32_t vr_id)
```

Input Parameters

`proc_id`	OSPF process identifier
`area_id`	Area identifier in an IPv4 address format
`val`	Translator stability interval in seconds <0-2147483647>
`vr_id`	Virtual router identifier; for a non-virtual-router implementation, specify 0

Output Parameters

None

Return Values

OSPF_API_SET_ERR_WRONG_VALUE when `val` is not in range

OSPF_API_SET_ERR_VR_NOT_EXIST when the virtual router is not found

OSPF_API_SET_ERR_PROCESS_NOT_EXIST when the given process does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF_API_SET_ERR_AREA_NOT_NSSA when the area is not an NSSA

OSPF_API_SET_SUCCESS when the function succeeds

CHAPTER 15 OSPFv3 SNMP API

This chapter describes the OSPFv3 management information base (MIB).

Overview of MIB Implementation

The `OSPF6-MIB.txt` file contains the MIB definitions for SNMP (based on the definitions in RFC 5643). The file is set up with the following sections:

- General Variables - Variables global to the OSPFv3 process. See General Variables.
- Area Table - Describes the OSPFv3 Areas that the router participates in. See Area Table.
- Area-Scope Link State Database - The LSDB contains the Area-Scope Link State Advertisements from throughout the area that the device is attached to.
- Link-Scope Link State Databases (non-virtual and virtual) - The LSDB contains the Link-Scope Link State Advertisements from the interfaces that the device is attached to. See Link-Scope Link State Database Table.
- AS-Scope Link State Database - The LSDB contains the AS-Scope Link State Advertisements from throughout the areas that the device is attached to.
- Host Table - Host/Metric table provides the configured Host route information. It indicates what hosts are directly attached to the router and their corresponding metric.
- Interface Table - Describes the various IPv6 links on which OSPFv3 is configured. See Interface Table.
- Virtual Interface Table - Describes the virtual interfaces that the OSPFv3 Process is configured. See Virtual Interface Table.
- Neighbor Table - Describes all neighbors in the locality of the OSPFv3 process. See Neighbor Table.
- Configured Neighbor Table - Describes all configured neighbors. See Configured Neighbor Table.
- Virtual Neighbor Table - Describes all virtual neighbors. See Virtual Neighbor Table.
- Area Aggregate Table - Describes the prefixes, which summarize routing information for export outside of an Area. See Area Aggregate Table.
- Virtual Link Lsdb Table - The LSDB contains the Link-Scope Link State Advertisements from virtual interfaces. See Virtual Link Lsdb Table.
- Notifications - These are generated when a entity undergoes a redundancy switchover.

General Variables

OSPFv3GeneralTable

Attribute	Syntax	Access	Function
ospfv3RouterId	IP address	read-write	ospf6_set_router_id
ospfv3AdminStat	INTEGER	read-write	ospf6_set_admin_stat
ospfv3ASBdrRtrStatus	INTEGER	read-write	ospf6_set_asbdr_rtr_status
ospfv3ReferenceBandwidth	INTEGER	read-write	ospf6_get_reference_bandwidth ospf6_set_reference_bandwidth
ospfv3RestartSupport	INTEGER	read-write	ospf6_get_restart_support ospf6_set_restart_support
ospfv3RestartInterval	INTEGER	read-write	ospf6_get_restart_interval ospf6_set_restart_interval
ospfv3RestartStrictLsaChecking	INTEGER	read-write	ospf6_get_restart_strict_lsa_check
ospfv3RestartStatus	INTEGER	read only	ospf6_get_restart_status
ospfv3RestartAge	INTEGER	read only	ospf6_get_restart_age
ospfv3RestartExitReason	INTEGER	read only	ospf6_get_restart_exit_reason
ospfv3NotificationEnable	INTEGER	read-write	ospf6_get_notification_enable
ospfv3StubRouterSupport	INTEGER	read only	ospf6_get_stub_router_support
ospfv3StubRouterAdvertisement	INTEGER	read-write	ospf6_get_stub_router_advertisement
ospfv3DiscontinuityTime	INTEGER	read only	ospf6_get_discontinuity_time
ospfv3RestartTime	INTEGER	read only	ospf6_get_restart_time

Area Table

OSPFv3 Area Table

Attribute	Syntax	Access	Function
ospfv3AreaSummary	INTEGER	read-create	ospf6_set_area_summary
ospfv3AreaStubMetricType	INTEGER	read-write	ospf6_get_area_stub_metric_type ospf6_get_next_area_stub_metric_type ospf6_set_area_stub_metric_type
ospfv3AreaTEEnabled	INTEGER	read-write	ospf6_get_area_te_enabled ospf6_get_next_area_te_enabled ospf6_set_area_te_enabled

Attribute	Syntax	Access	Function
ospfv3AreaNssaTranslatorRole	INTEGER	read-create	ospf6_set_area_nssa_trans_role
ospfv3AreaNssaTranslatorStabInterval	Unsigned32	read-create	ospf6_set_area_nssa_trans_stability_interval

Link-Scope Link State Database Table

OSPFv3LinkScopeLSDBTable

Attribute	Syntax	Access	Function
ospfv3LinkLsdbSequence	INTEGER	read only	ospf6_get_link_lsdb_sequence ospf6_get_next_link_lsdb_sequence
ospfv3LinkLsdbAge	INTEGER	read only	ospf6_get_link_lsdb_age ospf6_get_next_link_lsdb_age
ospfv3LinkLsdbChecksum	INTEGER	read only	ospf6_get_link_lsdb_checksum ospf6_get_next_link_lsdb_checksum
ospfv3LinkLsdbAdvertisement	STRING	read only	ospf6_get_link_lsdb_advertisement ospf6_get_next_link_lsdb_advertisement
ospfv3LinkLsdbTypeKnown	INTEGER,	read only	ospf6_get_link_lsdb_type_known ospf6_get_next_link_lsdb_type_known

Interface Table

OSPFv3IfTable

Attribute	Syntax	Access	Function
ospfv3IfDemandNbrProbe	INTEGER	read-write	ospf6_get_if_demand_nbr_probe ospf6_get_next_if_demand_nbr_probe
ospfv3IfDemandNbrProbeRetransLimit	INTEGER	read-write	ospf6_get_if_demand_nbr_probe_retrans_limit ospf6_get_next_if_demand_nbr_probe_retrans_limit
ospfv3IfDemandNbrProbeInterval	INTEGER	read-write	ospf6_get_if_demand_nbr_probe_interval ospf6_get_next_if_demand_nbr_probe_interval
ospfv3IfTEDisabled	INTEGER	read-write	ospf6_get_if_te_disabled ospf6_get_next_if_te_disabled
ospfv3IfLinkLSASuppression	INTEGER	read-write	ospf6_get_if_link_lsa_suppression ospf6_get_next_if_link_lsa_suppression ospf6_set_if_link_lsa_suppression
ospfv3IfAreaId	IP ADDRESS	read-write	ospf6_get_if_area_id ospf6_get_next_if_area_id

Attribute	Syntax	Access	Function
ospfv3IfType	INTEGER	read-write	ospf6_get_if_type ospf6_get_next_if_type ospf6_set_if_type
ospfv3IfAdminStatus	INTEGER	read-write	ospf6_get_if_admin_stat ospf6_get_next_if_admin_stat ospf6_set_if_admin_stat
ospfv3IfRtrPriority	INTEGER	read-write	ospf6_get_if_rtr_priority ospf6_get_next_if_rtr_priority ospf6_set_if_rtr_priority
ospfv3IfTransitDelay	INTEGER	read-write	ospf6_get_if_transit_delay ospf6_get_next_if_transit_delay ospf6_set_if_transit_delay
ospfv3IfRetransInterval	INTEGER	read-write	ospf6_get_if_retrans_interval ospf6_get_next_if_retrans_interval ospf6_set_if_retrans_interval
ospfv3IfHelloInterval	INTEGER	read-write	ospf6_get_if_hello_interval ospf6_get_next_if_hello_interval ospf6_set_if_hello_interval
ospfv3IfRtrDeadInterval	INTEGER	read-write	ospf6_get_if_rtr_dead_interval ospf6_get_next_if_rtr_dead_interval ospf6_set_if_rtr_dead_interval
ospfv3IfPollInterval	INTEGER	read-write	ospf6_get_if_poll_interval ospf6_get_next_if_poll_interval
ospfv3IfState	INTEGER	read-only	ospf6_get_if_state ospf6_get_next_if_state
ospfv3IfDesignatedRouter	IP ADDRESS	read-only	ospf6_get_if_dr ospf6_get_next_if_dr
ospfv3IfBackupDesignatedRouter	IP ADDRESS	read-only	ospf6_get_if_bdr ospf6_get_next_if_bdr
ospfv3IfEvents	Counter	read-only	ospf6_get_if_events ospf6_get_next_if_events
ospfv3IfRowStatus	INTEGER	read-write	ospf6_get_if_status ospf6_get_next_if_status
ospfv3IfDemand	INTEGER	read-write	ospf6_get_if_demand ospf6_get_next_if_demand
ospfv3IfMetricValue	INTEGER	read-write	ospf6_get_if_metric_value ospf6_get_next_if_metric_value ospf6_set_if_metric_value

Attribute	Syntax	Access	Function
ospfv3IfLinkScopeLsaCount	Gauge	read-only	ospf6_get_if_link_scope_lsa_count ospf6_get_next_if_link_scope_lsa_count
ospfv3IfLinkLsaCksumSum	INTEGER	read-only	ospf6_get_if_link_lsa_cksumsum ospf6_get_next_if_link_lsa_cksumsum

Virtual Interface Table

OSPFv3VirtIfTable

Attribute	Syntax	Access	Function
ospfv3VirtIfInstId	INTEGER	read only	ospf6_get_virt_if_instid ospf6_get_next_virt_if_instid

Neighbor Table

OSPFv3NbrTable

Attribute	Syntax	Access	Function
ospfv3NbrAddressType	INTEGER	read-only	ospf6_get_nbr_address_type ospf6_get_next_nbr_address_type
ospfv3NbrAddress	STRING	read-only	ospf6_get_nbr_address ospf6_get_next_nbr_address
ospfv3NbrOptions	INTEGER	read-only	ospf6_get_nbr_options ospf6_get_next_nbr_options
ospfv3NbrPriority	INTEGER	read-only	ospf6_get_nbr_priority ospf6_get_next_nbr_priority
ospfv3NbrState	INTEGER	read-only	ospf6_get_nbr_state ospf6_get_next_nbr_state
ospfv3NbrEvents	Counter	read-only	ospf6_get_nbr_events ospf6_get_next_nbr_events
ospfv3NbrLsRetransQLen	Gauge	read-only	ospf6_get_nbr_lsretransq_len ospf6_get_next_nbr_lsretransq_len
ospfv3NbrHelloSuppressed	INTEGER	read-only	ospf6_get_nbr_hello_suppressed ospf6_get_next_nbr_hello_suppressed
ospfv3NbrIfId	INTEGER	read-only	ospf6_get_nbr_if_id ospf6_get_next_nbr_if_id
ospfv3NbrRestartHelperStatus	INTEGER	read only	ospf6_get_nbr_restart_helper_status ospf6_get_next_nbr_restart_helper_status

Attribute	Syntax	Access	Function
ospfv3NbrRestartHelperAge	INTEGER	read only	ospf6_get_nbr_restart_helper_age ospf6_get_next_nbr_restart_helper_age
ospfv3NbrRestartHelperExitReason	INTEGER	read only	ospf6_get_nbr_restart_helper_exit_reason ospf6_get_next_nbr_restart_helper_exit_reason

Configured Neighbor Table

OSPFv3CfgNbrTable

Attribute	Syntax	Access	Function
ospfv3CfgNbrPriority	INTEGER	read-write	ospf6_get_cfg_nbr_priority ospf6_get_next_cfg_nbr_priority ospf6_set_cfg_nbr_priority
ospfv3CfgNbrRowStatus	INTEGER	read-write	ospf6_get_cfg_nbr_status ospf6_get_next_cfg_nbr_status ospf6_set_cfg_nbr_status

Virtual Neighbor Table

OSPFv3NbrTable

Attribute	Syntax	Access	Function
ospfv3VirtNbrIfInstId	INTEGER	read only	ospf6_get_virt_nbr_ifinstid ospf6_get_next_virt_nbr_ifinstid
ospfv3VirtNbrRestartHelperStatus	INTEGER	read only	ospf6_get_virt_nbr_restart_helper_status ospf6_get_next_virt_nbr_restart_helper_status
ospfv3VirtNbrRestartHelperAge	INTEGER	read only	ospf6_get_virt_nbr_restart_helper_age ospf6_get_next_virt_nbr_restart_helper_age
ospfv3VirtNbrRestartHelperExitReason	INTEGER	read only	ospf6_get_virt_nbr_restart_helper_exit_reason ospf6_get_next_virt_nbr_restart_helper_exit_reason

Area Aggregate Table

OSPFv3AreaAggregateTable

Attribute	Syntax	Access	Function
ospfv3AreaAggregateRouteTag	INTEGER	read-write	ospf6_get_area_aggregate_route_tag ospf6_get_next_area_aggregate_route_tag ospf6_set_area_aggregate_route_tag
ospfv3AreaAggregateEffect	INTEGER	read-write	ospf6_set_area_aggregate_effect
ospfv3AreaAggregateStatus	INTEGER	read-write	ospf6_set_area_aggregate_status

Virtual Link Lsdb Table

OSPFv3VirtLinkScopeLSDBTable

Attribute	Syntax	Access	Function
ospfv3VirtLinkLsdbSequence	INTEGER	read only	ospf6_get_virt_link_lsdb_sequence ospf6_get_next_virt_link_lsdb_sequence
ospfv3VirtLinkLsdbAge	INTEGER	read only	ospf6_get_virt_link_lsdb_age ospf6_get_next_virt_link_lsdb_age
ospfv3VirtLinkLsdbChecksum	INTEGER	read only	ospf6_get_virt_link_lsdb_checksum ospf6_get_next_virt_link_lsdb_checksum
ospfv3VirtLinkLsdbAdvertisement	STRING	read only	ospf6_get_virt_link_lsdb_advertisement ospf6_get_next_virt_link_lsdb_advertisement
ospfv3VirtLinkLsdbTypeKnown	INTEGER	read only	ospf6_get_virt_link_lsdb_type_known ospf6_get_next_virt_link_lsdb_type_known

Functions

Following is a summary of the OSPFv3 functions. Details are provided in the following subsections.

Function	Description
ospf6_get_area_aggregate_route_tag	Gets the aggregate route tag
ospf6_get_area_stub_metric_type	Gets the value of the metric type for the given area
ospf6_get_area_te_enabled	Gets the value that indicates whether traffic engineering is enabled for the given area
ospf6_get_cfg_nbr_priority	Gets the priority number of the neighbor in the Designated Router Election algorithm
ospf6_get_cfg_nbr_status	Gets the status of the given OSPFv3-configured neighbor
ospf6_get_discontinuity_time	Gets the value of sysUpTime counter
ospf6_get_if_admin_stat	Gets the administrative status of the given OSPFv3 interface
ospf6_get_if_area_id	Gets the area ID
ospf6_get_if_bdr	Gets the interface address of the Backup Designated Router (BDR)
ospf6_get_if_demand	Gets the value that indicates whether Demand OSPFv3 procedures are performed on the given interface
ospf6_get_if_demand_nbr_probe	Gets the value that indicates whether neighbor probing is enabled to determine whether the neighbor is inactive
ospf6_get_if_demand_nbr_probe_interval	Gets the number of times the neighbor is to be probed
ospf6_get_if_demand_nbr_probe_retrans_limit	Gets the configured number of consecutive LSA retransmissions permitted before the neighbor is considered inactive and the neighbor adjacency is brought down
ospf6_get_if_dr	Gets the router identifier of the Designated Router for the given OSPFv3 interface
ospf6_get_if_events	Gets the number of times the given OSPFv3 interface has changed its state or an error has occurred
ospf6_get_if_hello_interval	Gets the length of time, in seconds, between the Hello packets that the router sends on the given interface
ospf6_get_if_link_scope_lsa_count	Gets the total number of Link-Scope LSAs in the given link's LSDB
ospf6_get_if_link_lsa_cksumsum	Gets the 32-bit unsigned sum of the Link-Scope LSAs' LS checksums contained in the given link's LSDB
ospf6_get_if_link_lsa_suppression	Gets the value that indicates whether link LSA origination is suppressed for broadcast or NBMA interface types
ospf6_get_if_metric_value	Gets the metric value assigned to the given interface

Function	Description
ospf6_get_if_poll_interval	Gets the larger time interval, in seconds, between the Hello packets sent to an inactive, non-broadcast multi-access neighbor for the given interface
ospf6_get_if_retrans_interval	Gets the number of seconds between LSA retransmissions for adjacencies belonging to the given interface
ospf6_get_if_rtr_dead_interval	Gets the number of seconds that a router's Hello packets have not been seen before its neighbors declare the router down on the given interface
ospf6_get_if_rtr_priority	Gets the priority value of the given interface
ospf6_get_if_state	Gets the state of the given OSPFv3 interface
ospf6_get_if_status	Gets the status of whether OSPFv3 is configured on the given interface
ospf6_get_if_te_disabled	Gets the value that indicates whether traffic engineering is disabled on the given interface when traffic engineering is enabled in the area to which the interface is attached
ospf6_get_if_transit_delay	Gets the transit-delay value of the given OSPFv3 interface
ospf6_get_if_type	Gets the interface type of the given OSPFv3 interface
ospf6_get_link_lsdb_advertisement	Gets the entire LSA, including its header, from the local Link-Scope LSDB for the given interface
ospf6_get_link_lsdb_age	Gets the age of the LSA that is stored in the local Link-Scope LSDB
ospf6_get_link_lsdb_checksum	Gets the checksum of the contents of the LSA, excluding the age field, for the given OSPFv3 interface
ospf6_get_link_lsdb_sequence	Gets the LSDB sequence number of the local Link-Scope LSA for the given interface
ospf6_get_link_lsdb_type_known	Gets the value that indicates whether the LSA type is recognized by the given router
ospf6_get_nbr_address	Gets the IPv6 address of the neighbor associated with the local link
ospf6_get_nbr_address_type	Gets the address type of the ospfv3NbrAddress object
ospf6_get_nbr_events	Gets the number of times the relationship with the neighbor has changed state, or an error has occurred
ospf6_get_nbr_hello_suppressed	Gets the value that indicates whether hello packets are suppressed
ospf6_get_nbr_if_id	Gets the interface identifier that the neighbor advertises in its Hello Packets on this link (or on the local interface index of the neighbor)
ospf6_get_nbr_lsretransq_len	Gets the current length of the retransmission queue for a neighbor
ospf6_get_nbr_options	Gets the bit mask corresponding to the neighbor's options field
ospf6_get_nbr_priority	Gets the priority number of the neighbor
ospf6_get_nbr_restart_helper_age	Gets the remaining time in the current OSPF graceful restart interval for the given router that is acting as a restart helper for the neighbor

Function	Description
ospf6_get_nbr_restart_helper_exit_reason	Gets the outcome of the last attempt the given router was acting as a graceful restart helper for the neighbor
ospf6_get_nbr_restart_helper_status	Gets the value that indicates whether the given router is a graceful restart helper for the neighbor
ospf6_get_nbr_state	Gets the state of the relationship of the given router with the neighbor
ospf6_get_next_area_aggregate_route_tag	Gets the next entry for the aggregate route tag
ospf6_get_next_area_stub_metric_type	Gets the next entry for the value of the metric type for the given area
ospf6_get_next_area_te_enabled	Gets the next entry for the value that indicates whether traffic engineering is enabled for the given area
ospf6_get_next_cfg_nbr_priority	Gets the next entry for the priority number of the neighbor in the Designated Router Election algorithm
ospf6_get_next_cfg_nbr_status	Gets the next entry for the status of the given OSPFv3-configured neighbor
ospf6_get_next_if_admin_stat	Gets the next entry for the administrative status of the given OSPFv3 interface
ospf6_get_next_if_area_id	Gets the next entry for the area ID
ospf6_get_next_if_bdr	Gets the next entry for the interface address of the BDR
ospf6_get_next_if_demand	Gets the next entry of the value that indicates whether Demand OSPFv3 procedures are performed on the given interface
ospf6_get_next_if_demand_nbr_probe	Gets the next entry for the value that indicates whether neighbor probing is enabled to determine whether the neighbor is inactive
ospf6_get_next_if_demand_nbr_probe_interval	Gets the number of times the neighbor is to be probed
ospf6_get_next_if_demand_nbr_probe_retrans_limit	Gets the next value for the configured number of consecutive LSA retransmissions permitted before the neighbor is considered inactive and the neighbor adjacency is brought down
ospf6_get_next_if_dr	Gets the next entry for the router identifier of the Designated Router for the given OSPFv3 interface
ospf6_get_next_if_events	Gets the next entry for the number of times the OSPFv3 interface has changed its state or an error has occurred
ospf6_get_next_if_hello_interval	Gets the next entry for the value of the interface hello interval
ospf6_get_next_if_link_lsa_cksumsum	Gets the next entry for the 32-bit unsigned sum of the Link-Scope LSAs' LS checksums contained in the given link's LSDB
ospf6_get_next_if_link_lsa_suppression	Gets the next entry for the value that indicates whether the link LSA origination is suppressed for broadcast or NBMA interface types
ospf6_get_next_if_link_scope_lsa_count	Gets the next entry value for the total number of Link-Scope LSAs for the given interface in this link's LSDB
ospf6_get_next_if_metric_value	Gets the next entry for the metric value assigned to the given interface

Function	Description
ospf6_get_next_if_poll_interval	Gets the next entry of the larger time interval, in seconds, between the Hello packets sent to an inactive, non-broadcast multi-access neighbor on the given interface
ospf6_get_next_if_retrans_interval	Gets the next entry for the number of seconds between LSA retransmissions for adjacencies belonging to the given interface
ospf6_get_next_if_rtr_dead_interval	Gets the next entry for the number of seconds that the router's Hello packets have not been seen before its neighbors declare the router down on the given interface
ospf6_get_next_if_rtr_priority	Gets the next entry for the priority value of the given interface
ospf6_get_next_if_state	Gets the next entry value for the state of the given OSPFv3 interface
ospf6_get_next_if_status	Gets the next entry value for the status of whether OSPFv3 is configured on the given interface
ospf6_get_next_if_te_disabled	Gets the next entry value that indicates whether traffic engineering is disabled on the given interface when traffic engineering is enabled in the area to which the interface is attached
ospf6_get_next_if_transit_delay	Gets the next entry for the transit-delay value of the given OSPFv3 interface
ospf6_get_next_if_type	Gets the next entry value for the interface type of the given OSPFv3 interface
ospf6_get_next_link_lsdb_advertisement	Gets the next entry value for the LSA, including its header, from the local Link-Scope LSDB for the given interface
ospf6_get_next_link_lsdb_age	Gets the next entry value for the age of the LSA that is stored in local Link-Scope LSDB
ospf6_get_next_link_lsdb_checksum	Gets the next entry value for the checksum of the contents of the LSA that is stored in the local Link-Scope LSDB, excluding the age field, for the given interface
ospf6_get_next_link_lsdb_sequence	Gets the next entry value for the LSDB sequence number of the local Link-Scope LSA for the given interface
ospf6_get_next_link_lsdb_type_known	Gets the next entry value that indicates whether the LSA type is recognized by the given router
ospf6_get_next_nbr_address	Gets the next value for the IPv6 address of the neighbor associated with the local link
ospf6_get_next_nbr_address_type	Gets the next value for the address type of the ospfv3NbrAddress object
ospf6_get_next_nbr_events	Gets the next value for the number of times the relationship with the neighbor has changed state, or an error has occurred
ospf6_get_next_nbr_hello_suppressed	Gets the next value that indicates whether hello packets are suppressed
ospf6_get_next_nbr_if_id	Gets the next value for the interface identifier that the neighbor advertises in its Hello Packets on this link

Function	Description
ospf6_get_next_nbr_lsretransq_len	Gets the next value for the current length of the retransmission queue for a neighbor
ospf6_get_next_nbr_options	Gets next value for the bit mask corresponding to the neighbor's options field
ospf6_get_next_nbr_priority	Gets the next value for the priority number of the neighbor
ospf6_get_next_nbr_restart_helper_age	Gets the next value for the remaining time in the current OSPF graceful restart interval for the given router that is acting as a restart helper for the neighbor
ospf6_get_next_nbr_restart_helper_exit_reason	Gets the next value for the outcome of the last attempt the given router was acting as a graceful restart helper for the neighbor
ospf6_get_next_nbr_restart_helper_status	Gets the next value that indicates whether the given router is a graceful restart helper for the neighbor
ospf6_get_next_nbr_state	Gets the next value for the state of the relationship of the given router with the neighbor
ospf6_get_next_virt_if_instid	Gets the next entry value for the local Interface Instance identifier assigned by the OSPFv3 process to the given OSPFv3 virtual interface
ospf6_get_next_virt_link_lsdb_advertisement	Gets the next entry value for the LSA, including its header, from the local Link-Scope LSDB for the OSPFv3 virtual link
ospf6_get_next_virt_link_lsdb_age	Gets the next entry for the age of the LSA that is stored in the local Link-Scope LSDB for the OSPFv3 virtual link
ospf6_get_next_virt_link_lsdb_checksum	Gets the next entry for the checksum of the contents of the LSA, excluding the age field, for the OSPFv3 virtual link
ospf6_get_next_virt_link_lsdb_sequence	Gets the next entry for the LSDB sequence number of the local Link-Scope LSA for the OSPFv3 virtual link
ospf6_get_next_virt_link_lsdb_type_known	Gets the next entry for the value that indicates whether the LSA type is recognized by the given router
ospf6_get_next_virt_nbr_ifinstid	Gets the next entry for the interface instance identifier of the given virtual interface over which the neighbor can be reached
ospf6_get_next_virt_nbr_restart_helper_age	Gets the next entry for the value of the remaining time in the current OSPF graceful restart interval for the given virtual router that is acting as a restart helper for the neighbor
ospf6_get_next_virt_nbr_restart_helper_exit_reason	Gets the next entry for the outcome of the last attempt the given virtual router was acting as a graceful restart helper for the neighbor
ospf6_get_next_virt_nbr_restart_helper_status	Gets the next entry for the outcome of the last attempt the given virtual router was acting as a graceful restart helper for the neighbor
ospf6_get_notification_enable	Gets the value that indicates whether the generation of OSPFv3 notifications is enabled
ospf6_get_reference_bandwidth	Gets the reference bandwidth in kilobits per second (Kbps)

Function	Description
ospf6_get_restart_age	Gets remaining time, in seconds, in the current OSPF graceful restart interval for the router
ospf6_get_restart_exit_reason	Gets the outcome of the last attempt at a graceful restart for the router
ospf6_get_restart_interval	Gets the interval of the graceful restart timeout for the router
ospf6_get_restart_status	Gets the current status of OSPF graceful restart capability for the router
ospf6_get_restart_strict_lsa_check	Gets the value that indicates whether strict LSA checking is enabled for graceful restart on the router
ospf6_get_restart_support	Gets the value that indicates whether the router supports OSPF graceful restart
ospf6_get_restart_time	Gets the value of sysUpTime on the most recent time at which the ospfv3RestartExitReason object was updated
ospf6_get_stub_router_advertisement	Gets the value that indicates whether the router advertises stub or standard LSAs
ospf6_get_stub_router_support	Gets the value that indicates whether the router supports the stub router functionality
ospf6_get_virt_if_instid	Gets the local Interface Instance identifier assigned by the OSPFv3 process to the given OSPFv3 virtual interface
ospf6_get_virt_link_lsdb_advertisement	Gets the entire LSA, including its header, from the local Link-Scope LSDB for the OSPFv3 virtual link
ospf6_get_virt_link_lsdb_age	Gets the age of the LSA that is stored in the local Link-Scope LSDB for the given virtual OSPFv3 interface
ospf6_get_virt_link_lsdb_checksum	Gets the checksum of the contents of the LSA, excluding the age field, for the given virtual OSPFv3 interface
ospf6_get_virt_link_lsdb_sequence	Gets the LSDB sequence number of the local Link-Scope LSA for the given virtual interface
ospf6_get_virt_link_lsdb_type_known	Gets the value that indicates whether the LSA type is recognized by the given router
ospf6_get_virt_nbr_ifinstid	Gets the interface instance identifier of the given virtual interface over which the neighbor can be reached
ospf6_get_virt_nbr_restart_helper_age	Gets the remaining time in the current OSPF graceful restart interval for the given virtual router that is acting as a restart helper for the neighbor
ospf6_get_virt_nbr_restart_helper_exit_reason	Gets the outcome of the last attempt of the given virtual router at acting as a graceful restart helper for the neighbor
ospf6_get_virt_nbr_restart_helper_status	Gets the value that indicates whether the virtual router is acting as a graceful restart helper for the neighbor
ospf6_set_admin_stat	Sets the administrative status of the OSPFv3 process in the router
ospf6_set_area_aggregate_effect	Sets or unsets the advertisement of the aggregate

Function	Description
ospf6_set_area_aggregate_route_tag	Sets or unsets the aggregate route tag
ospf6_set_area_aggregate_status	Sets the status of whether management of the table is permitted by facilitating actions, such as row creation, construction, and destruction
ospf6_set_area_nssa_trans_role	Sets the value that indicates an NSSA border router's policy to perform NSSA translation of NSSA-LSAs into AS-External-LSAs
ospf6_set_area_summary	Sets a value that controls the import of Inter-Area summary LSAs into stub areas
ospf6_set_area_nssa_trans_stability_interval	Sets the stability interval for the NSSA area
ospf6_set_area_stub_metric_type	Sets the type of metric advertised as a default route for the given area
ospf6_set_area_te_enabled	Sets the value that indicates whether traffic engineering is enabled for the given area
ospf6_set_asbdr_rtr_status	Sets the router as an Autonomous System (AS) border router
ospf6_set_cfg_nbr_priority	Sets the priority number of this neighbor in the Designated Router Election algorithm
ospf6_set_cfg_nbr_status	Sets the status of the given OSPFv3-configured neighbor
ospf6_set_if_admin_stat	Sets the administrative status of the given OSPFv3 interface
ospf6_set_if_hello_interval	Sets the length of time, in seconds, between the hello packets that the router sends on the given interface
ospf6_set_if_link_lsa_suppression	Sets the value used to specify whether link LSA origination is suppressed for broadcast or NBMA interface types
ospf6_set_if_metric_value	Sets the metric assigned to the given interface
ospf6_set_if_retrans_interval	Sets the duration in seconds between LSA retransmissions for adjacencies belonging to the given interface
ospf6_set_if_rtr_dead_interval	Sets the number of seconds that a router's Hello packets have not been seen on the given interface before its neighbors declare the router down
ospf6_set_if_rtr_priority	Sets the priority value of the given interface
ospf6_set_if_transit_delay	Sets the transit-delay value of the given OSPFv3 interface
ospf6_set_if_type	Sets the interface type of the given OSPFv3 interface
ospf6_get_reference_bandwidth	Gets the reference bandwidth in Kbps
ospf6_set_restart_interval	Sets the interval for the graceful restart timeout for the configured OSPFv3
ospf6_get_restart_support	Gets the value that indicates whether the router supports OSPF graceful restart

Function	Description
ospf6_set_router_id	Sets the router identifier, which uniquely identifies the router in the Autonomous System
ospf6_set_stub_metric	Sets the metric value advertised for the default route into Stub and NSSA areas

ospf6_get_area_aggregate_route_tag

This function gets the aggregate route tag, which is advertised only in the summarized As-External LSA when summarizing from NSSA-LSAs to AS-External-LSAs.

Syntax

```
int
ospf6_get_area_aggregate_route_tag (char *tag, struct pal_in4_addr area_id,
                                    int area_lsdb_type, int prefix_type,
                                    struct pal_in6_addr prefix, int prefix_len,
                                    int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`area_lsdb_type`	Area LSDB type that the given Address Aggregate applies to
`prefix_type`	Prefix type of ospfv3AreaAggregatePrefix object
`prefix`	IPv6 prefix
`prefix_len`	Length of the prefix (in bits). A prefix can not be shorter than 3 bits.

Output Parameters

`ret`	Aggregate route tag

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_area_stub_metric_type

This function gets the value of the metric type for the given area. This value is advertised for the default route into stub and NSSA areas. By default, this equals the least metric among the interfaces to other areas.

Syntax

```
int
ospf6_get_area_stub_metric_type (char *tag, struct pal_in4_addr area_id,
                                 int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format

Output Parameters

`ret`	Metric type:
`1`	OSPFv3 Metric
`2`	External Type 1 (comparableCost)
`3`	External Type 2 (Noncomparable)

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_area_te_enabled

This function gets the value that indicates whether traffic engineering is enabled for the given area.

Syntax

```
int
ospf6_get_area_te_enabled (char *tag, struct pal_in4_addr area_id,
                           int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format

Output Parameters

`ret`	Traffic engineering status:
`OSPF6_API_TRUE`	Enabled
`OSPF6_API_FALSE`	Disabled

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_cfg_nbr_priority

This function gets the priority number of the neighbor in the Designated Router Election algorithm. This number is used in the Designated Router Election algorithm to determine whether the neighbor is eligible to become the Designated Router on a given network. The value 0 disqualifies the router.

Syntax

```
int
ospf6_get_cfg_nbr_priority (char *tag, int ifindex, int instid,
                            int nbr_addr_type, struct pal_in6_addr nbr_addr,
                            int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor

Output Parameters

`ret`	Priority number of the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_cfg_nbr_status

This function gets the status of the given OSPFv3-configured neighbor.

Syntax

```
int
ospf6_get_cfg_nbr_status (char *tag, int ifindex, int instid,
                          int nbr_addr_type, struct pal_in6_addr nbr_addr,
                          int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor

Output Parameters

`ret`	Status of the OSPFv3-configured neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_discontinuity_time

This function gets the value of sysUpTime counter, which is the most recent time at which any one of this MIB's counters experienced a discontinuity. If no such discontinuities occurred since the last re-initialization of the local management subsystem, then the sysUpTime counter is set to zero.

Syntax

```
int
ospf6_get_discontinuity_time (char *tag, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

Output Parameters

`ret`	Value of sysUpTime counter

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_admin_stat

This function gets the administrative status of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_if_admin_stat (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

Output Parameters

`ret` Administrative status of the OSPFv3 interface:

`OSPF6_API_STATUS_ENABLED`
Enabled

`OSPF6_API_STATUS_DISABLED`
Disabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_area_id

This function gets the area ID, which is a 32-bit integer uniquely identifying the area to which the given interface connects. Area ID 0 is used for the OSPFv3 backbone.

Syntax

```
int
ospf6_get_if_area_id (char *tag, int ifindex, int instid,
                      struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Area identifier

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_bdr

This function gets the interface address of the Backup Designated Router (BDR).

Syntax

```
int
ospf6_get_if_bdr (char *tag, int ifindex, int instid,
                  struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Interface address of the BDR

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_demand

This function gets the value that indicates whether Demand OSPFv3 procedures (Hello suppression to FULL neighbors and setting the DoNotAge flag on propagated LSAs) are performed on the given interface.

Syntax

```
int
ospf6_get_if_demand (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Value indicating whether Demand OSPFv3 procedures are performed on the given interface:
`OSPF6_API_FALSE`	Disabled. This is the default value.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_demand_nbr_probe

This function gets the value that indicates whether neighbor probing is enabled to determine whether the neighbor is inactive. If neighbor probing is disabled, the neighbor is inactive.

Syntax

```
int
ospf6_get_if_demand_nbr_probe (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Neighbor probe value:
`OSPF6_API_FALSE`	Disabled. This is the default value.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_demand_nbr_probe_interval

This function gets the number of times the neighbor is to be probed.

Syntax

```
int
ospf6_get_if_demand_nbr_probe_interval (char *tag, int ifindex,
                                        int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Neighbor probe interval. The default value is 120.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_demand_nbr_probe_retrans_limit

This function gets the configured number of consecutive LSA retransmissions permitted before the neighbor is considered inactive and the neighbor adjacency is brought down.

Syntax

```
int
ospf6_get_if_demand_nbr_probe_retrans_limit (char *tag, int ifindex,
                                             int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Number of consecutive LSA retransmissions permitted. The default value is 10.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_dr

This function gets the router identifier of the Designated Router for the given OSPFv3 interface.

Syntax

```
int
ospf6_get_if_dr (char *tag, int ifindex, int instid,
                 struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index of this OSPFv3 interface
`instid`	Instance identifier

Output Parameters

`ret`	Router identifier of the Designated Router

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_events

This function gets the number of times the given OSPFv3 interface has changed its state or an error has occurred.

Syntax

```
int
ospf6_get_if_events (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index of this OSPFv3 interface
`instid`	Instance identifier

Output Parameters

`ret`	Number of events

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_hello_interval

This function gets the length of time, in seconds, between the Hello packets that the router sends on the given interface. This value must be the same for all routers attached to a common network.

Syntax

```
int
ospf6_get_if_hello_interval (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Interface hello interval in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_link_scope_lsa_count

This function gets the total number of Link-Scope LSAs in the given link's LSDB.

Syntax

```
int
ospf6_get_if_link_scope_lsa_count (char *tag, int ifindex, int instid,
                                   int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Total number of Link-Scope LSAs

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_link_lsa_cksumsum

This function gets the 32-bit unsigned sum of the Link-Scope LSAs' LS checksums contained in the given link's LSDB.

Syntax

```
int
ospf6_get_if_link_lsa_cksumsum (char *tag, int ifindex, int instid,
                                int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	The 32-bit unsigned sum of the Link-Scope LSAs' LS checksums

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_link_lsa_suppression

This function gets the value that indicates whether link LSA origination is suppressed for broadcast or NBMA interface types.

Syntax

```
int
ospf6_get_if_link_lsa_suppression (char *tag, int ifindex,
                                   int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret` Suppression value for the Link LSA origination:

`OSPF6_API_FALSE`
: Not Suppressed

`OSPF6_API_TRUE`
: Suppressed

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_metric_value

This function gets the metric value assigned to the given interface.

Syntax

```
int
ospf6_get_if_metric_value (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Metric value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_poll_interval

This function gets the larger time interval, in seconds, between the Hello packets sent to an inactive, non-broadcast multi-access neighbor for the given interface.

Syntax

```
int
ospf6_get_if_poll_interval (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Poll interval for the interface:
`OSPF_POLL_INTERVAL_DEFAULT`	Default value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_retrans_interval

This function gets the number of seconds between LSA retransmissions for adjacencies belonging to the given interface.

Syntax

```
int
ospf6_get_if_retrans_interval (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Interval between LSA retransmissions

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_rtr_dead_interval

This function gets the number of seconds that a router's Hello packets have not been seen before its neighbors declare the router down on the given interface.

Syntax

```
int
ospf6_get_if_rtr_dead_interval (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Router dead interval, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_rtr_priority

This function gets the priority value of the given interface.

A value of 0 signifies that the router is not eligible to become the Designated Router on this particular network. In the event of a tie in this value, the routers use their Router identifier as a tie breaker.

Syntax

```
int
ospf6_get_if_rtr_priority (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	Interface priority value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_state

This function gets the state of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_if_state (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

Output Parameters

`ret`	State of the OSPFv3 interface:
`1`	Interface is down
`2`	Interface is a loopback interface
`3`	Interface is in a waiting state
`4`	Interface is a point-to-point interface
`5`	Interface is a Designated Router
`6`	Interface is a Backup Designated Router

7	Interface is an Other Designated Router
8	Interface is in a standby state

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_status

This function gets the status of whether OSPFv3 is configured on the given interface.

Syntax

```
int
ospf6_get_if_status (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

tag	OSPFv3 process tag
ifindex	Interface index
instid	Instance identifier

Output Parameters

ret	Row status of the interface:
ROW_STATUS_ACTIVE	Row is active
ROW_STATUS_NONEXISTENT	Row is nonexistent

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_te_disabled

This function gets the value that indicates whether traffic engineering is disabled on the given interface when traffic engineering is enabled in the area to which the interface is attached.

Syntax

```
int
ospf6_get_if_te_disabled (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

tag	OSPFv3 process tag
ifindex	Interface index
instid	Instance identifier

Output Parameters

`ret` Status of traffic engineering:

`OSPF6_API_FALSE`
Enabled. This is the default value.

`OSPF6_API_TRUE`
Disabled.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_transit_delay

This function gets the transit-delay value of the given OSPFv3 interface. This value is an estimate of the number of seconds required to transmit a link-state update packet through the interface.

Syntax

```
int
ospf6_get_if_transit_delay (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

Output Parameters

`ret` Transit-delay value, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_if_type

This function gets the interface type of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_if_type (char *tag, int ifindex, int instid, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

Output Parameters

`ret`	Interface types:
`1`	Broadcast
`2`	Non-broadcast multi-access (NBMA)
`3`	Point-to-Point
`5`	Point-to-Multipoint

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_link_lsdb_advertisement

This function gets the entire LSA, including its header, from the local Link-Scope LSDB for the given interface.

Syntax

```
ospf6_get_link_lsdb_advertisement (char *tag, int ifindex, int instid,
                                   int type, struct pal_in4_addr router_id,
                                   struct pal_in4_addr ls_id,
                                   u_char **ret, size_t *size);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Entire LSA including its header
`size`	Length of the LSA content

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_link_lsdb_age

This function gets the age of the LSA that is stored in the local Link-Scope LSDB.

Syntax

```
int
ospf6_get_link_lsdb_age (char *tag, int ifindex, int instid, int type,
```

```
                        struct pal_in4_addr router_id,
                        struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Age of the LSA, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_link_lsdb_checksum

This function gets the checksum of the contents of the LSA that is stored in the specified local Link-Scope LSDB, excluding the age field, for the given OSPFv3 interface.

Syntax

```
int
ospf6_get_link_lsdb_checksum (char *tag, int ifindex, int instid, int type,
                              struct pal_in4_addr router_id,
                              struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Checksum of the contents of the LSA excluding the age field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_link_lsdb_sequence

This function gets the LSDB sequence number of the local Link-Scope LSA for the given interface. The sequence number is used to detect old and duplicate LSAs.

Syntax

```
int
ospf6_get_link_lsdb_sequence (char *tag, int ifindex, int instid,
                              int type, struct pal_in4_addr router_id,
                              struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	LSDB sequence number of the LSA

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_link_lsdb_type_known

This function gets the value that indicates whether the LSA type is recognized by the given router.

Syntax

```
int
ospf6_get_link_lsdb_type_known (char *tag, int ifindex, int instid,
                                int type, struct pal_in4_addr router_id,
                                struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret` Status of whether the LSA type is recognized:

`OSPF6_API_FALSE`
Not recognized

`OSPF6_API_TRUE`
Recognized

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_address

This function gets the IPv6 address of the neighbor associated with the local link.

Syntax

```
int
ospf6_get_nbr_address (char *tag, int ifindex, int instid,
                       struct pal_in4_addr router_id,
                       u_char **ret, size_t *size);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	IPv6 address of OSPFv3 neighbor
`size`	Length of the IPv6 address of the OSPFv3 neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_address_type

This function gets the address type of the ospfv3NbrAddress object.

Syntax

```
int
ospf6_get_nbr_address_type (char *tag, int ifindex, int instid,
                            struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Address type of the ospfv3NbrAddress object:
`INET_ADDRESS_TYPE_IPV6`	An IPv6 address

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_events

This function gets the number of times the relationship with the neighbor has changed state, or an error has occurred.

Syntax

```
int
ospf6_get_nbr_events (char *tag, int ifindex, int instid,
                      struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Number of events for the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_hello_suppressed

This function gets the value that indicates whether hello packets are suppressed.

Syntax

```
int
ospf6_get_nbr_hello_suppressed (char *tag, int ifindex, int instid,
                                struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Status of whether hello packets are suppressed:
`OSPF6_API_FALSE`	Not suppressed

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_if_id

This function gets the interface identifier that the neighbor advertises in its Hello Packets on this link (or on the local interface index of the neighbor).

Syntax

```
int
ospf6_get_nbr_if_id (char *tag, int ifindex, int instid,
                     struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Interface identifier that the neighbor advertises

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_lsretransq_len

This function gets the current length of the retransmission queue for a neighbor.

Syntax

```
int
ospf6_get_nbr_lsretransq_len (char *tag, int ifindex, int instid,
```

```
                        struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Length of the retransmission queue

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_options

This function gets the bit mask corresponding to the neighbor's options field.

Syntax

```
int
ospf6_get_nbr_options (char *tag, int ifindex, int instid,
                       struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Bit mask corresponding to the neighbor's options field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_priority

This function gets the priority number of the neighbor. This number is used in the Designated Router Election algorithm to determine whether the neighbor is eligible to become the Designated Router on a given network. The value 0 disqualifies the router.

Syntax

```
int
ospf6_get_nbr_priority (char *tag, int ifindex, int instid,
```

```
                        struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Priority number of the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_restart_helper_age

This function gets the remaining time in the current OSPF graceful restart interval for the given router that is acting as a restart helper for the neighbor.

Syntax

```
int
ospf6_get_nbr_restart_helper_age (char *tag, int ifindex, int instid,
                                  struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor

Output Parameters

`ret`	Remaining time, in seconds, in the current OSPF graceful restart interval

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_restart_helper_exit_reason

This function gets the outcome of the last attempt the given router was acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_nbr_restart_helper_exit_reason (char *tag, int ifindex, int instid,
```

```
                                        struct pal_in4_addr router_id,
                                        int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor

Output Parameters

`ret`	Exit reason:
`none`	No restart has yet been attempted
`inProgress`	A restart attempt is currently underway
`completed`	No restart has yet been attempted
`timedOut`	Last time restart timed out
`topologyChanged`	Last time restart was aborted due to a topology change

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_restart_helper_status

This function gets the value that indicates whether the given router is a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_nbr_restart_helper_status (char *tag, int ifindex, int instid,
                                     struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor

Output Parameters

`ret`	Neighbor restart helper status:
`OSPF6_API_RESTART_HELPER`	Graceful restart helper
`SPF6_API_RESTART_HELPER_NONE`	Not a graceful restart helper

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_nbr_state

This function gets the state of the relationship of the given router with the neighbor.

Syntax

```
int
ospf6_get_nbr_state (char *tag, int ifindex, int instid,
                     struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	State of the relationship with this neighbor:
`1`	down
`2`	attempt
`3`	init
`4`	TwoWay
`5`	exchangeStart
`6`	exchange
`7`	loading
`8`	full

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_area_aggregate_route_tag

This function gets the next entry for the aggregate route tag, which is advertised only in the summarized As-External LSA when summarizing from NSSA-LSAs to AS-External-LSAs.

Syntax

```
int
ospf6_get_next_area_aggregate_route_tag (char *tag,
                                         struct pal_in4_addr *area_id,
                                         int *area_lsdb_type, int *prefix_type,
```

```
                                        struct pal_in6_addr *prefix,
                                        int *prefix_len, int indexlen,
                                        int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`area_lsdb_type`	Area LSDB type that the given Address Aggregate applies to
`prefix_type`	Prefix type of ospfv3AreaAggregatePrefix object
`prefix`	IPv6 prefix
`prefix_len`	Length of the prefix (in bits). A prefix can not be shorter than 3 bits
`index_len`	Length of the index

Output Parameters

`ret`	Aggregate route tag

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_area_stub_metric_type

This function gets the next entry for the value of the metric type for the given area. This value is advertised for the default route into stub and NSSA areas. By default, this equals the least metric among the interfaces to other areas.

Syntax

```
int
ospf6_get_next_area_stub_metric_type (char *tag, struct pal_in4_addr *area_id,
                                      int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Length of the object index

Output Parameters

`ret`	Metric type:
`1`	OSPFv3 Metric
`2`	External Type 1 (comparableCost)
`3`	External Type 2 (Noncomparable)

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_area_te_enabled

This function gets the next entry for the value that indicates whether traffic engineering is enabled for the given area.

Syntax

```
int
ospf6_get_next_area_te_enabled (char *tag, struct pal_in4_addr *area_id,
                                int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`indexlen`	Length of the object index

Output Parameters

`ret` Traffic engineering status:

`OSPF6_API_FALSE`
: Disabled

`OSPF6_API_TRUE`
: Enabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_cfg_nbr_priority

This function gets the next entry for the priority number of the neighbor in the Designated Router Election algorithm. This number is used in the Designated Router Election algorithm to determine whether the neighbor is eligible to become the Designated Router on a given network. The value 0 disqualifies the router.

Syntax

```
int
ospf6_get_next_cfg_nbr_priority (char *tag, int *ifindex, int *instid,
                                 int *nbr_addr_type,
                                 struct pal_in6_addr *nbr_addr,
                                 int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor

`indexlen`	Length of the object index

Output Parameters

`ret`	Priority number of the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_cfg_nbr_status

This function gets the next entry for the status of the given OSPFv3-configured neighbor.

Syntax

```
int
ospf6_get_next_cfg_nbr_status (char *tag, int *ifindex, int *instid,
                               int *nbr_addr_type,
                               struct pal_in6_addr *nbr_addr,
                               int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor
`indexlen`	Length of the object index

Output Parameters

`ret`	Status of the OSPFv3-configured neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_admin_stat

This function gets the next entry for the administrative status of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_next_if_admin_stat (char *tag, int *ifindex, int *instid,
                              int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Administrative status of the OSPFv3 interface:
OSPF6_API_STATUS_ENABLED	Enabled
OSPF6_API_STATUS_DISABLED	Disabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_area_id

This function gets the next entry for the area ID. This ID is a 32-bit integer uniquely identifying the area to which the given interface connects. Area ID 0 is used for the OSPFv3 backbone.

Syntax

```
int
ospf6_get_next_if_area_id (char *tag, int *ifindex, int *instid,
                           int indexlen, struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Area identifier

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_bdr

This function gets the next entry for the interface address of the BDR.

Syntax

```
int
ospf6_get_next_if_bdr (char *tag, int *ifindex, int *instid,
```

```
                    int indexlen, struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface address of the BDR

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_demand

This function gets the next entry of the value that indicates whether Demand OSPFv3 procedures (Hello suppression to FULL neighbors and setting the DoNotAge flag on propagated LSAs) are performed on the given interface.

Syntax

```
int
ospf6_get_next_if_demand (char *tag, int *ifindex, int *instid,
                          int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Value indicating whether Demand OSPFv3 procedures are performed on the given interface:

`OSPF6_API_FALSE`

Disabled. This is the default value.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_demand_nbr_probe

This function gets the next entry for the value that indicates whether neighbor probing is enabled to determine whether the neighbor is inactive. Neighbor probing is disabled by default.

Syntax

```
int
ospf6_get_next_if_demand_nbr_probe (char *tag, int *ifindex, int * instid,
                                    int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Neighbor probe value:
`OSPF6_API_FALSE`	Disabled. This is the default value.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_demand_nbr_probe_interval

This function gets the number of times the neighbor is to be probed. The default value is 120.

Syntax

```
int
ospf6_get_next_if_demand_nbr_probe_interval (char *tag, int *ifindex,
                                             int *instid, int indexlen,
                                             int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Neighbor probe interval

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_demand_nbr_probe_retrans_limit

This function gets the next value for the configured number of consecutive LSA retransmissions permitted before the neighbor is considered inactive and the neighbor adjacency is brought down.

Syntax

```
int
ospf6_get_next_if_demand_nbr_probe_retrans_limit (char *tag, int *ifindex,
                                                  int *instid, int indexlen,
                                                  int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	The number of consecutive LSA retransmissions permitted. The default value is 10.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_dr

This function gets the next entry for the router identifier of the Designated Router for the given OSPFv3 interface.

Syntax

```
int
ospf6_get_next_if_dr (char *tag, int *ifindex, int *instid, int indexlen,
                      struct pal_in4_addr *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	The next router identifier of the Designated Router of OSPFv3 interface

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_events

This function gets the next entry for the number of times the OSPFv3 interface has changed its state or an error has occurred.

Syntax

```
int
ospf6_get_next_if_events (char *tag, int *ifindex, int *instid,
                          int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Number of events

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_hello_interval

This function gets the next entry for the value of the interface hello interval. This is the length of time, in seconds, between the Hello packets that the router sends on the given interface. This value must be the same for all routers attached to a common network.

Syntax

```
int
ospf6_get_next_if_hello_interval (char *tag, int *ifindex, int *instid,
                                  int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface hello interval in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_link_lsa_cksumsum

This function gets the next entry for the 32-bit unsigned sum of the Link-Scope LSAs' LS checksums contained in the given link's LSDB.

Syntax

```
int
ospf6_get_next_if_link_lsa_cksumsum (char *tag, int *ifindex, int *instid,
                                     int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	The 32-bit unsigned sum of the Link-Scope LSAs' LS checksums

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_link_lsa_suppression

This function gets the next entry for the value that indicates whether the link LSA origination is suppressed for broadcast or NBMA interface types.

Syntax

```
int
ospf6_get_next_if_link_lsa_suppression (char *tag, int *ifindex, int *instid,
                                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Suppression value for the Link LSA origination:

`OSPF6_API_FALSE`
: Not Suppressed

`OSPF6_API_TRUE`
: Suppressed

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_link_scope_lsa_count

This function gets the next entry value for the total number of Link-Scope LSAs for the given interface in this link's LSDB.

Syntax

```
int
ospf6_get_next_if_link_scope_lsa_count (char *tag, int *ifindex, int *instid,
                                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index of this OSPFv3 interface
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	The total number of Link-Scope LSAs

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_metric_value

This function gets the next entry for the metric value assigned to the given interface.

Syntax

```
int
ospf6_get_next_if_metric_value (char *tag, int *ifindex, int *instid,
                                int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

`indexlen` Length of the object index

Output Parameters

`ret` Metric value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_poll_interval

This function gets the next entry of the larger time interval, in seconds, between the Hello packets sent to an inactive, non-broadcast multi-access neighbor on the given interface.

Syntax

```
int
ospf6_get_next_if_poll_interval (char *tag, int *ifindex, int *instid,
                                 int indexlen, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

`indexlen` Length of the object index

Output Parameters

`ret` Poll interval for the given interface

`OSPF_POLL_INTERVAL_DEFAULT`

Default poll interval value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_retrans_interval

This function gets the next entry for the number of seconds between LSA retransmissions for adjacencies belonging to the given interface.

Syntax

```
int
ospf6_get_next_if_retrans_interval (char *tag, int *ifindex, int *instid,
                                    int indexlen, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Interval between LSA retransmissions

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_rtr_dead_interval

This function gets the next entry for the number of seconds that the router's Hello packets have not been seen before its neighbors declare the router down on the given interface.

Syntax

```
int
ospf6_get_next_if_rtr_dead_interval (char *tag, int *ifindex, int *instid,
                                     int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Router dead interval, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_rtr_priority

This function gets the next entry for the priority value of the given interface.

A value of 0 signifies that the router is not eligible to become the Designated Router on this particular network. In the event of a tie in this value, the routers use their Router identifier as a tie breaker.

Syntax

```
int
ospf6_get_next_if_rtr_priority (char *tag, int *ifindex, int *instid,
                                int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface priority value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_state

This function gets the next entry value for the state of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_next_if_state (char *tag, int *ifindex, int *instid,
                         int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	State of the OSPFv3 interface:
`1`	Interface is down
`2`	Interface is a loopback interface
`3`	Interface is in a waiting state
`4`	Interface is a point-to-point interface
`5`	Interface is a Designated Router
`6`	Interface is a Backup Designated Router
`7`	Interface is an Other Designated Router
`8`	Interface is in a standby state

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_status

This function gets the next entry value for the status of whether OSPFv3 is configured on the given interface.

Syntax

```
int
ospf6_get_next_if_status (char *tag, int *ifindex, int *instid,
                          int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret` Row status of the given interface:

`ROW_STATUS_ACTIVE`

Row is active

`ROW_STATUS_NONEXISTENT`

Row is nonexistent

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_te_disabled

This function gets the next entry value that indicates whether traffic engineering is disabled on the given interface when traffic engineering is enabled in the area to which the interface is attached.

Syntax

```
int
ospf6_get_next_if_te_disabled (char *tag, int *ifindex, int *instid,
                               int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret` Status of traffic engineering:

`OSPF6_API_FALSE`

Enabled. This is the default value.

`OSPF6_API_TRUE`

Disabled.

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_transit_delay

This function gets the next entry for the transit-delay value of the given OSPFv3 interface. This value is an estimate of the number of seconds required to transmit a link-state update packet through the interface.

Syntax

```
int
ospf6_get_next_if_transit_delay (char *tag, int *ifindex, int *instid,
                                 int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Transit-delay value, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_if_type

This function gets the next entry value for the interface type of the given OSPFv3 interface.

Syntax

```
int
ospf6_get_next_if_type (char *tag, int *ifindex, int *instid,
                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface types:
`1`	Broadcast
`2`	Non-broadcast multi-access (NBMA)
`3`	Point-to-Point
`5`	Point-to-Multipoint

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_link_lsdb_advertisement

This function gets the next entry value for the LSA, including its header, from the local Link-Scope LSDB for the given interface.

Syntax

```
int
ospf6_get_next_link_lsdb_advertisement (char *tag, int *ifindex, int *instid,
                                        int *type, struct pal_in4_addr *router_id,
                                        struct pal_in4_addr *ls_id,
                                        int indexlen, u_char **ret,
                                        size_t *size);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	The entire LSA including its header
`size`	The length of the LSA content

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_link_lsdb_age

This function gets the next entry value for the age of the LSA that is stored in local Link-Scope LSDB.

Syntax

```
int
ospf6_get_next_link_lsdb_age (char *tag, int *ifindex, int *instid,
                              int *type, struct pal_in4_addr *router_id,
                              struct pal_in4_addr *ls_id,
                              int indexlen, int *ret);
```

Input Parameters

tag	OSPFv3 process tag
ifindex	Identifier of the link from which the LSA was received
instid	Instance identifier
type	Type of LSA that is stored in the local Link-Scope LSDB
router_id	Router identifier of the originating router
ls_id	Link state identifier of the LSA
indexlen	Length of the object index

Output Parameters

ret	Age of the LSA, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_link_lsdb_checksum

This function gets the next entry value for the checksum of the contents of the LSA that is stored in the local Link-Scope LSDB, excluding the age field, for the given interface.

Syntax

```
int
ospf6_get_next_link_lsdb_checksum (char *tag, int *ifindex, int *instid,
                                   int *type, struct pal_in4_addr *router_id,
                                   struct pal_in4_addr *ls_id,
                                   int indexlen, int *ret);
```

Input Parameters

tag	OSPFv3 process tag
ifindex	Identifier of the link from which the LSA was received
instid	Instance identifier
type	Type of LSA that is stored in the local Link-Scope LSDB
router_id	Router identifier of the originating router

`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret`	Checksum of the contents of the LSA excluding the age field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_link_lsdb_sequence

This function gets the next entry value for the LSDB sequence number of the local Link-Scope LSA for the given interface. The sequence number is used to detect old and duplicate LSAs.

Syntax

```
int
ospf6_get_next_link_lsdb_sequence (char *tag, int *ifindex, int *instid,
                                   int *type, struct pal_in4_addr *router_id,
                                   struct pal_in4_addr *ls_id,
                                   int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret`	LSDB sequence number of the LSA

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_link_lsdb_type_known

This function gets the next entry value that indicates whether the LSA type is recognized by the given router.

Syntax

```
int
ospf6_get_next_link_lsdb_type_known (char *tag, int *ifindex, int *instid,
                                     int *type, struct pal_in4_addr *router_id,
```

```
                                    struct pal_in4_addr *ls_id,
                                    int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Identifier of the link from which the LSA was received
`instid`	Instance identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret` Status of whether the LSA type is recognized:

`OSPF6_API_FALSE`
: Not recognized

`OSPF6_API_TRUE`
: Recognized

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_address

This function gets the next value for the IPv6 address of the neighbor associated with the local link.

Syntax

```
int
ospf6_get_next_nbr_address (char *tag, int *ifindex, int *instid,
                            struct pal_in4_addr *router_id,
                            int indexlen, u_char **ret, size_t *size);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	IPv6 address of OSPFv3 neighbor
`size`	Length of the IPv6 address of the OSPFv3 neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_address_type

This function gets the next value for the address type of the ospfv3NbrAddress object.

Syntax

```
int
ospf6_get_next_nbr_address_type (char *tag, int *ifindex, int *instid,
                                 struct pal_in4_addr *router_id,
                                 int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Address type of the ospfv3NbrAddress object:
`INET_ADDRESS_TYPE_IPV6`	An IPv6 address

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_events

This function gets the next value for the number of times the relationship with the neighbor has changed state, or an error has occurred.

Syntax

```
int
ospf6_get_next_nbr_events (char *tag, int *ifindex, int *instid,
                           struct pal_in4_addr *router_id,
                           int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier

`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Number of events for the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_hello_suppressed

This function gets the next value that indicates whether hello packets are suppressed.

Syntax

```
int
ospf6_get_next_nbr_hello_suppressed (char *tag, int *ifindex, int *instid,
                                     struct pal_in4_addr *router_id,
                                     int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Status of whether hello packets are suppressed:
`OSPF6_API_FALSE`	Not suppressed

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_if_id

This function gets the next value for the interface identifier that the neighbor advertises in its Hello Packets on this link (or on the local interface index of the neighbor).

Syntax

```
int
ospf6_get_next_nbr_if_id (char *tag, int *ifindex, int *instid,
                          struct pal_in4_addr *router_id,
                          int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface identifier that the neighbor advertises

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_lsretransq_len

This function gets the next value for the current length of the retransmission queue for a neighbor.

Syntax

```
int
ospf6_get_next_nbr_lsretransq_len (char *tag, int *ifindex, int *instid,
                                   struct pal_in4_addr *router_id,
                                   int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Length of the retransmission queue

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_options

This function gets next value for the bit mask corresponding to the neighbor's options field.

Syntax

```
int
ospf6_get_next_nbr_options (char *tag, int *ifindex, int *instid,
```

```
                                struct pal_in4_addr *router_id,
                                int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Bit mask corresponding to the neighbor's options field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_priority

This function gets the next value for the priority number of the neighbor. This number is used in the Designated Router Election algorithm to determine whether the neighbor is eligible to become the Designated Router on a given network. The value 0 disqualifies the router.

Syntax

```
int
ospf6_get_next_nbr_priority (char *tag, int *ifindex, int *instid,
                             struct pal_in4_addr *router_id,
                             int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Priority number of the neighbor

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_restart_helper_age

This function gets the next value for the remaining time in the current OSPF graceful restart interval for the given router that is acting as a restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_nbr_restart_helper_age (char *tag, int *ifindex, int *instid,
                                       struct pal_in4_addr *router_id,
                                       int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor
`indexlen`	Length of the object index

Output Parameters

`ret`	Remaining time, in seconds, in the current OSPF graceful restart interval

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_restart_helper_exit_reason

This function gets the next value for the outcome of the last attempt the given router was acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_nbr_restart_helper_exit_reason (char *tag, int *ifindex,
                                               int *instid,
                                               struct pal_in4_addr *router_id,
                                               int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor
`indexlen`	Length of the object index

Output Parameters

`ret`	Exit reason:

`none` No restart has yet been attempted

`inProgress` A restart attempt is currently underway

`completed` No restart has yet been attempted

`timedOut` Last time restart timed out

`topologyChanged`
Last time restart was aborted due to a topology change

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_restart_helper_status

This function gets the next value that indicates whether the given router is a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_nbr_restart_helper_status (char *tag, int *ifindex, int *instid,
                                          struct pal_in4_addr *router_id,
                                          int indexlen, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

`router_id` Router identifier of the neighbor

`indexlen` Length of the object index

Output Parameters

`ret` Neighbor restart helper status:

`OSPF6_API_RESTART_HELPER`
Graceful restart helper

`SPF6_API_RESTART_HELPER_NONE`
Not a graceful restart helper

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_nbr_state

This function gets the next value for the state of the relationship of the given router with the neighbor.

Syntax

```
int
ospf6_get_next_nbr_state (char *tag, int *ifindex, int *instid,
                          struct pal_in4_addr *router_id,
                          int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`router_id`	Router identifier of the neighbor
`indexlen`	Length of the object index

Output Parameters

`ret`	State of the relationship with this neighbor:
`1`	down
`2`	attempt
`3`	init
`4`	TwoWay
`5`	exchangeStart
`6`	exchange
`7`	loading
`8`	full

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_if_instid

This function gets the next entry value for the local Interface Instance identifier assigned by the OSPFv3 process to the given OSPFv3 virtual interface.

Syntax

```
int
ospf6_get_next_virt_if_instid (char *tag, struct pal_in4_addr *area_id,
                               struct pal_in4_addr *router_id,
                               int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

ret	Local Interface Instance identifier of the OSPFv3 virtual interface

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_link_lsdb_advertisement

This function gets the next entry value for the LSA, including its header, from the local Link-Scope LSDB for the OSPFv3 virtual link.

Syntax

```
int
ospf6_get_next_virt_link_lsdb_advertisement (char *tag,
                                             struct pal_in4_addr *area_id,
                                             struct pal_in4_addr *nbr_id,
                                             int *type,
                                             struct pal_in4_addr *router_id,
                                             struct pal_in4_addr *ls_id,
                                             int indexlen, u_char **ret,
                                             size_t *size);
```

Input Parameters

tag	OSPFv3 process tag
area_id	Area identifier in an IPv4 address format
nbr_id	Neighbor process identifier
type	Type of LSA that is stored in the local Link-Scope LSDB
router_id	Router identifier of the originating router
ls_id	Link state identifier of the LSA
indexlen	Length of the object index

Output Parameters

ret	Entire LSA including its header
size	Length of the LSA content

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_link_lsdb_age

This function gets the next entry for the age of the LSA that is stored in the local Link-Scope LSDB for the OSPFv3 virtual link.

Syntax

```
int
ospf6_get_next_virt_link_lsdb_age (char *tag, struct pal_in4_addr *area_id,
                                   struct pal_in4_addr *nbr_id, int *type,
                                   struct pal_in4_addr *router_id,
                                   struct pal_in4_addr *ls_id, int indexlen,
                                   int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret`	Age of the LSA, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_link_lsdb_checksum

This function gets the next entry for the checksum of the contents of the LSA that is stored in the specified local Link-Scope LSDB, excluding the age field, for the OSPFv3 virtual link.

Syntax

```
int
ospf6_get_next_virt_link_lsdb_checksum (char *tag,
                                        struct pal_in4_addr *area_id,
                                        struct pal_in4_addr *nbr_id, int *type,
                                        struct pal_in4_addr *router_id,
                                        struct pal_in4_addr *ls_id,
                                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

`indexlen`	Length of the object index

Output Parameters

`ret`	Checksum of the complete contents of the LSA excluding the age field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_link_lsdb_sequence

This function gets the next entry for the LSDB sequence number of the local Link-Scope LSA for the OSPFv3 virtual link. The sequence number is used to detect old and duplicate LSAs.

Syntax

```
int
ospf6_get_next_virt_link_lsdb_sequence (char *tag,
                                        struct pal_in4_addr *area_id,
                                        struct pal_in4_addr *nbr_id, int *type,
                                        struct pal_in4_addr *router_id,
                                        struct pal_in4_addr *ls_id,
                                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret`	LSDB sequence number of the LSA

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_link_lsdb_type_known

This function gets the next entry for the value that indicates whether the LSA type is recognized by the given router.

Syntax

```
int
ospf6_get_next_virt_link_lsdb_type_known (char *tag,
```

```
                                        struct pal_in4_addr *area_id,
                                        struct pal_in4_addr *nbr_id,
                                        int *type,
                                        struct pal_in4_addr *router_id,
                                        struct pal_in4_addr *ls_id,
                                        int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA
`indexlen`	Length of the object index

Output Parameters

`ret` Status of whether the LSA type is recognized:

`OSPF6_API_FALSE`
: Not recognized

`OSPF6_API_TRUE`
: Recognized

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_next_virt_nbr_ifinstid

This function gets the next entry for the interface instance identifier of the given virtual interface over which the neighbor can be reached.

Syntax

```
int
ospf6_get_next_virt_nbr_ifinstid (char *tag, struct pal_in4_addr *area_id,
                                  struct pal_in4_addr *router_id,
                                  int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Interface instance identifier

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_next_virt_nbr_restart_helper_age

This function gets the next entry for the value of the remaining time in the current OSPF graceful restart interval for the given virtual router that is acting as a restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_virt_nbr_restart_helper_age (char *tag,
                                            struct pal_in4_addr *area_id,
                                            struct pal_in4_addr *router_id,
                                            int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router
`indexlen`	Length of the object index

Output Parameters

`ret`	Remaining time, in seconds, in the current OSPF graceful restart interval

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_next_virt_nbr_restart_helper_exit_reason

This function gets the next entry for the outcome of the last attempt the given virtual router was acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_virt_nbr_restart_helper_exit_reason (char *tag,
                                                    struct pal_in4_addr *area_id,
                                                    struct pal_in4_addr *router_id,
                                                    int indexlen, int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag

`area_id` Area identifier in an IPv4 address format

`router_id` Router identifier of the originating router

`indexlen` Length of the object index

Output Parameters

`ret` Exit reason:

- `none` No restart has yet been attempted
- `inProgress` A restart attempt is currently underway
- `completed` No restart has yet been attempted
- `timedOut` Last time restart timed out
- `topologyChanged` Last time restart was aborted due to a topology change

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_next_virt_nbr_restart_helper_status

This function gets the next entry for the outcome of the last attempt the given virtual router was acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_next_virt_nbr_restart_helper_status (char *tag,
                                               struct pal_in4_addr *area_id,
                                               struct pal_in4_addr *router_id,
                                               int indexlen, int *ret);
```

Input Parameters

`tag` OSPFv3 Process tag

`area_id` Area identifier in an IPv4 address format

`router_id` Router identifier of the originating router

`indexlen` Length of the object index

Output Parameters

`ret` Neighbor restart helper status:

- `OSPF6_API_RESTART_HELPER` Graceful restart helper
- `SPF6_API_RESTART_HELPER_NONE` Not a graceful restart helper

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_notification_enable

This function gets the value that indicates whether the generation of OSPFv3 notifications is enabled.

Syntax

```
int
ospf6_get_notification_enable (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Status of whether the generation of OSPFv3 notifications is enabled:

`OSPF6_API_FALSE`
Disabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_reference_bandwidth

This function gets the reference bandwidth in Kbps. The reference value is used for calculating the default interface metric. The default value is 100,000 Kbps.

Syntax

```
int
ospf6_get_reference_bandwidth (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Reference bandwidth in Kbps

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_age

This function gets remaining time, in seconds, in the current OSPF graceful restart interval for the router.

Syntax

```
int
ospf6_get_restart_age (char *tag, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

Output Parameters

`ret`	Remaining time, in seconds, in the current OSPF graceful restart interval

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_exit_reason

This function gets the outcome of the last attempt at a graceful restart for the router.

Syntax

```
int
ospf6_get_restart_exit_reason (char *tag, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

Output Parameters

`ret`	Exit reason:
`none`	No restart has yet been attempted
`inProgress`	A restart attempt is currently underway
`completed`	Last restart completed successfully
`timedOut`	Last time restart timed out
`topologyChanged`	Last time restart was aborted due to a topology change

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_interval

This function gets the interval of the graceful restart timeout for the router.

Syntax

```
int
ospf6_get_restart_interval (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Interval of the graceful restart timeout

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_status

This function gets the current status of OSPF graceful restart capability for the router.

Syntax

```
int
ospf6_get_restart_status (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Status of OSPF graceful restart:

`OSPF6_API_RESTART_UNPLANNED`
Unplanned

`OSPF6_API_RESTART_PLANNED`
Planned

`OSPF6_API_RESTART_NONE`
None

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_strict_lsa_check

This function gets the value that indicates whether strict LSA checking is enabled for graceful restart on the router.

Syntax

```
int
ospf6_get_restart_strict_lsa_check (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Status of whether strict LSA checking is enabled:

`OSPF6_API_RESTART_LSA_CHECK_ENABLE`
Enabled

`OSPF6_API_RESTART_LSA_CHECK_DISABLE`
Disabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_support

This function gets the value that indicates whether the router supports OSPF graceful restart.

Syntax

```
int
ospf6_get_restart_support (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Status of support for OSPF graceful restart:

`1` None

`3` Planned and Unplanned Restart

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_restart_time

This function gets the value of sysUpTime on the most recent time at which the ospfv3RestartExitReason object was updated.

Syntax

```
int
ospf6_get_restart_time (char *tag, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

Output Parameters

`ret`	Value of sysUpTime

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_stub_router_advertisement

This function gets the value that indicates whether the router advertises stub or standard LSAs.

Syntax

```
int
ospf6_get_stub_router_advertisement (char *tag, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag

Output Parameters

`ret`	Type of stub router advertisements:
`1`	Standard LSAs; the default value

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_stub_router_support

This function gets the value that indicates whether the router supports the stub router functionality.

Syntax

```
int
ospf6_get_stub_router_support (char *tag, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

Output Parameters

`ret` Status for support of stub router functionality:

`OSPF6_API_FALSE`

Disabled

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_if_instid

This function gets the local Interface Instance identifier assigned by the OSPFv3 process to the given OSPFv3 virtual interface.

Syntax

```
int
ospf6_get_virt_if_instid (char *tag, struct pal_in4_addr area_id,
                          struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag` OSPFv3 process tag

`area_id` Area identifier in an IPv4 address format

`router_id` Router identifier of the originating router

Output Parameters

`ret` Local Interface Instance identifier of the OSPFv3 virtual interface

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_link_lsdb_advertisement

This function gets the entire LSA, including its header, from the local Link-Scope LSDB for the OSPFv3 virtual link.

Syntax

```
int
ospf6_get_virt_link_lsdb_advertisement (char *tag, struct pal_in4_addr area_id,
                                        struct pal_in4_addr nbr_id, int type,
                                        struct pal_in4_addr router_id,
                                        struct pal_in4_addr ls_id,
                                        u_char **ret, size_t *size);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Entire LSA including its header
`size`	Length of the LSA content

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_link_lsdb_age

This function gets the age of the LSA that is stored in the local Link-Scope LSDB for the given virtual OSPFv3 interface.

Syntax

```
int
ospf6_get_virt_link_lsdb_age (char *tag, struct pal_in4_addr area_id,
                              struct pal_in4_addr nbr_id, int type,
                              struct pal_in4_addr router_id,
                              struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Age of the LSA, in seconds

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_link_lsdb_checksum

This function gets the checksum of the contents of the LSA that is stored in the specified local Link-Scope LSDB, excluding the age field, for the given virtual OSPFv3 interface.

Syntax

```
int
ospf6_get_virt_link_lsdb_checksum (char *tag, struct pal_in4_addr area_id,
                                   struct pal_in4_addr nbr_id, int type,
                                   struct pal_in4_addr router_id,
                                   struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	Checksum of the complete contents of the LSA excluding the age field

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_link_lsdb_sequence

This function gets the LSDB sequence number of the local Link-Scope LSA for the given virtual interface. The sequence number is used to detect old and duplicate LSAs.

Syntax

```
int ospf6_get_virt_link_lsdb_sequence (char *tag, struct pal_in4_addr area_id,
                                       struct pal_in4_addr nbr_id, int type,
                                       struct pal_in4_addr router_id,
                                       struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret`	LSDB sequence number of the LSA

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_link_lsdb_type_known

This function gets the value that indicates whether the LSA type is recognized by the given router.

Syntax

```
int
ospf6_get_virt_link_lsdb_type_known (char *tag, struct pal_in4_addr area_id,
                                     struct pal_in4_addr nbr_id, int type,
                                     struct pal_in4_addr router_id,
                                     struct pal_in4_addr ls_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`nbr_id`	Neighbor process identifier
`type`	Type of LSA that is stored in the local Link-Scope LSDB
`router_id`	Router identifier of the originating router
`ls_id`	Link state identifier of the LSA

Output Parameters

`ret` Status of whether the LSA type is recognized:

`OSPF6_API_FALSE`
: Not recognized

`OSPF6_API_TRUE`
: Recognized

Return Values

OSPF6_API_GET_ERROR when the function fails

OSPF6_API_GET_SUCCESS when the function succeeds

ospf6_get_virt_nbr_ifinstid

This function gets the interface instance identifier of the given virtual interface over which the neighbor can be reached.

Syntax

```
int
ospf6_get_virt_nbr_ifinstid (char *tag, struct pal_in4_addr area_id,
                             struct pal_in4_addr router_id, int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Interface instance identifier

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_virt_nbr_restart_helper_age

This function gets the remaining time in the current OSPF graceful restart interval for the given virtual router that is acting as a restart helper for the neighbor.

Syntax

```
int
ospf6_get_virt_nbr_restart_helper_age (char *tag,
                                       struct pal_in4_addr area_id,
                                       struct pal_in4_addr router_id,
                                       int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Remaining time, in seconds, in the current OSPF graceful restart interval

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_virt_nbr_restart_helper_exit_reason

This function gets the outcome of the last attempt of the given virtual router at acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_virt_nbr_restart_helper_exit_reason (char *tag,
                                               struct pal_in4_addr area_id,
                                               struct pal_in4_addr router_id,
```

```
                                                int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Exit reason:
`none`	No restart has yet been attempted
`inProgress`	A restart attempt is currently underway
`completed`	No restart has yet been attempted
`timedOut`	Last time restart timed out
`topologyChanged`	Last time restart was aborted due to a topology change

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_get_virt_nbr_restart_helper_status

This function gets the value that indicates whether the virtual router is acting as a graceful restart helper for the neighbor.

Syntax

```
int
ospf6_get_virt_nbr_restart_helper_status (char *tag,
                                          struct pal_in4_addr area_id,
                                          struct pal_in4_addr router_id,
                                          int *ret);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`router_id`	Router identifier of the originating router

Output Parameters

`ret`	Neighbor restart helper status:
`OSPF6_API_RESTART_HELPER`	Graceful restart helper
`SPF6_API_RESTART_HELPER_NONE`	Not a graceful restart helper

Return Values

OSPF6_API_GET_SUCCESS when the function succeeds

OSPF6_API_GET_ERROR when the function fails

ospf6_set_admin_stat

This function sets the administrative status of the OSPFv3 process in the router.

Syntax

```
int
ospf6_set_admin_stat (char *tag, int status);
```

Input Parameters

`tag` OSPFv3 Process tag

`status` Administrative status:

OSPF6_API_STATUS_ENABLED
Enabled

OSPF6_API_STATUS_DISABLED
Disabled

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_CANT_INITIATE when the process cannot initiate

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPF process does not exist

ospf6_set_area_aggregate_effect

This function sets or unsets the advertisement of the aggregate. Prefixes subsumed by ranges will either trigger the advertisement of the indicated aggregate (advertiseMatching) or result in the prefix not being advertised at all outside the area.

Syntax

```
int
ospf6_set_area_aggregate_effect (char *tag, struct pal_in4_addr area_id,
                                 int area_lsdb_type, int prefix_type,
                                 struct pal_in6_addr prefix,
                                 int prefix_len, u_int32_t effect);
```

Input Parameters

`tag` OSPFv3 process tag

`area_id`	Area identifier in an IPv4 address format
`area_lsdb_type`	Area LSDB type that the given Address Aggregate applies to
`prefix_type`	Prefix type of ospfv3AreaAggregatePrefix object
`prefix`	IPv6 prefix
`prefix_len`	Length of the prefix (in bits). A prefix can not be shorter than 3 bits.
`effect`	Advertisement of the aggregate: OSPF6_API_AREA_RANGE_ADVERTISE_MATCHING Advertise OSPF6_API_AREA_RANGE_DONT_ADVERTISE_MATCHING Do not Advertise

Output Parameters

None

Return Values

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

ospf6_set_area_aggregate_route_tag

This function sets or unsets the aggregate route tag, which is advertised only in the summarized As-External LSA when summarizing from NSSA-LSAs to AS-External-LSAs.

Syntax

```
int
ospf6_set_area_aggregate_route_tag (char *tag, struct pal_in4_addr area_id,
                                    int area_lsdb_type, int prefix_type,
                                    struct pal_in6_addr prefix,
                                    int prefix_len, u_int32_t route_tag);
```

Input Parameters

`tag`	OSPFv3 process tag
`area_id`	Area identifier in an IPv4 address format
`area_lsdb_type`	Area LSDB type that the given Address Aggregate applies to
`prefix_type`	Prefix type of ospfv3AreaAggregatePrefix object
`prefix`	IPv6 prefix
`prefix_len`	Length of the prefix (in bits). A prefix can not be shorter than 3 bits.
`route_tag`	Aggregate route tag

Output Parameters

None

Return Values

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_EXIST when the same summary address exists

ospf6_set_area_aggregate_status

This function sets the status of whether management of the table is permitted by facilitating actions, such as row creation, construction, and destruction.

Syntax

```
iint
ospf6_set_area_aggregate_status (char *tag, struct pal_in4_addr area_id,
                                 int area_lsdb_type, int prefix_type,
                                 struct pal_in6_addr prefix,
                                 int prefix_len, u_int32_t status);
```

Input Parameters

`tag` OSPFv3 process tag

`area_id` Area identifier in an IPv4 address format

`area_lsdb_type` Area LSDB type that the given Address Aggregate applies to

`prefix_type` Prefix type of ospfv3AreaAggregatePrefix object

`prefix` IPv6 prefix

`prefix_len` Length of the prefix (in bits). A prefix can not be shorter than 3 bits.

`status` Status of aggregate:

ROW_STATUS_ACTIVE
: Active

ROW_STATUS_NOTINSERVICE
: Not in service

ROW_STATUS_NOTREADY
: Not ready

ROW_STATUS_CREATEANDGO
: Create active row

ROW_STATUS_CREATEANDWAIT
: Create row in wait state

ROW_STATUS_DESTROY
: Delete the row

Output Parameters

None

Return Values

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the specified process does not exist

OSPF6_API_SET_ERR_SUMMARY_ADDRESS_NOT_EXIST when the summary address does not exist

ospf6_set_area_nssa_trans_role

This function sets the value that indicates an NSSA border router's policy to perform NSSA translation of NSSA-LSAs into AS-External-LSAs.

Syntax

```
int
ospf6_set_area_nssa_trans_role (char *tag, struct pal_in4_addr area_id,
                                int role);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`role`	Translation role of NSSA border router

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

ospf6_set_area_nssa_trans_stability_interval

This function sets the stability interval for the NSSA area. This interval is the number of seconds after an elected translator determines its services are no longer required that it should continue to perform its translation duties.

Syntax

```
int
ospf6_set_area_nssa_trans_stability_interval (char *tag,
                                              struct pal_in4_addr area_id,
                                              int interval);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`interval`	Stability interval for the NSSA area, in seconds

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

ospf6_set_area_stub_metric_type

This function sets the type of metric advertised as a default route for the given area. This value is advertised for the default route into stub and NSSA areas. By default, this equals the least metric among the interfaces to other areas.

Syntax

```
int
ospf6_set_area_stub_metric_type (char *tag, struct pal_in4_addr area_id,
                                 int mtype);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`mtype`	Metric type

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_IS_BACKBONE when the specified area is the backbone

OSPF6_API_SET_ERR_METRIC_TYPE_INVALID when the specified metric type is not valid

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF6_API_SET_ERR_AREA_NOT_NSSA when the specified area is not an NSSA

ospf6_set_area_summary

This function sets a value that controls the import of Inter-Area summary LSAs into stub areas.

Syntax

```
int
ospf6_set_area_summary (char *tag, struct pal_in4_addr area_id, int val);
```

Input Parameters

`tag` OSPFv3 Process tag

`area_id` Area identifier in an IPv4 address format

`val` Area value; one of these constants from `ospf6d/ospf6_api.h`:

`OSPF6_NO_AREA_SUMMARY`
No area summary

`OSPF6_SEND_AREA_SUMMARY`
Send area summary

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when the area is default

ospf6_set_area_te_enabled

This function sets the value that indicates whether traffic engineering is enabled for the given area.

Syntax

```
int
ospf6_set_area_te_enabled (char *tag, struct pal_in4_addr area_id,
                           int status);
```

Input Parameters

`tag` OSPFv3 process tag

`area_id` Area identifier in an IPv4 address format

`status` Traffic engineering status

OSPF6_API_STATUS_ENABLED
Enabled

OSPF6_API_STATUS_DISABLED
Disabled

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

ospf6_set_asbdr_rtr_status

This function sets the router as an Autonomous System (AS) border router.

Syntax

```
int
ospf6_set_asbdr_rtr_status (char *tag, int status);
```

Input Parameters

`tag` OSPFv3 Process tag

`status` Router status:

`OSPF6_API_TRUE`
AS border router

`OSPF6_API_FALSE`
Not an AS border router

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

ospf6_set_cfg_nbr_priority

This function sets the priority number of this neighbor in the Designated Router Election algorithm. The value 0 signifies that the neighbor is not eligible to become the Designated Router on this particular network.

Syntax

```
int
ospf6_set_cfg_nbr_priority (char *tag, int ifindex, int instid,
                            int nbr_addr_type, struct pal_in6_addr nbr_addr,
                            int priority);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor
`priority`	Priority number of the neighbor

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_ERR_NBR_CONFIG_INVALID when the neighbor command is allowed only on NBMA and point-to-multipoint networks

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_NBR_NOT_EXIST when the neighbor does not exist

ospf6_set_cfg_nbr_status

This function sets the status of the given OSPFv3-configured neighbor.

Syntax

```
int
ospf6_set_cfg_nbr_status (char *tag, int ifindex, int instid,
                          int nbr_addr_type, struct pal_in6_addr nbr_addr,
                          int status);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`nbr_addr_type`	Neighbor address type
`nbr_addr`	IP address of the neighbor
`status`	Status of the OSPFv3-configured neighbor

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_ERR_NBR_CONFIG_INVALID when the neighbor command is allowed only on NBMA and point-to-multipoint networks

OSPF6_API_SET_ERR_INCONSISTENT_VALUE when the neighbor row status is an inconsistent or incorrect value

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_NBR_NOT_EXIST when the neighbor does not exist

ospf6_set_if_admin_stat

This function sets the administrative status of the given OSPFv3 interface.

Syntax

```
int
ospf6_set_if_admin_stat (char *tag, int ifindex,
                         int instid, int adminstat);
```

Input Parameters

`tag` OSPFv3 process tag

`ifindex` Interface index

`instid` Instance identifier

`adminstat` Administrative status of the OSPFv3 interface:

`OSPF6_API_STATUS_ENABLED`
Enabled

`OSPF6_API_STATUS_DISABLED`
Disabled

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_AREA_ID_FORMAT_INVALID when the area identifier format is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_AREA when the interface is owned by another area

OSPF6_API_SET_ERR_IF_OWNED_BY_OTHER_PROCESS when the interface is owned by another process

OSPF6_API_SET_ERR_IF_INSTANCE_ID_MISMATCH when an interface instance identifier mismatch occurs

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the interface instance identifier is invalid

OSPF6_API_SET_ERR_IF_NO_LINKLOCAL_ADDRESS when there is no link local address

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

OSPF6_API_SET_ERR_IF_NOT_EXIST when the interface does not exist

ospf6_set_if_hello_interval

This function sets the length of time, in seconds, between the hello packets that the router sends on the given interface. This value must be the same for all routers attached to a common network.

Syntax

```
int
ospf6_set_if_hello_interval (char *tag, int ifindex,
                             int instid, int interval);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`interval`	Interface hello interval in seconds <1–65335>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_HELLO_INTERVAL_INVALID when the hello interval is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_link_lsa_suppression

This functions sets the value used to specify whether link LSA origination is suppressed for broadcast or NBMA interface types.

Syntax

```
int
ospf6_set_if_link_lsa_suppression (char *tag, int ifindex,
                                   int instid, int lsa_value);
```

Input Parameters

`tag`	OSPFv3 process tag

`ifindex`	Interface index
`instid`	Instance identifier
`lsa_value`	Suppression value for the Link LSA origination:
`OSPF6_API_FALSE`	Not Suppressed; this is the default value
`OSPF6_API_TRUE`	Suppressed

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_IF_PARAM_NOT_CONFIGURED when the interface parameter is not configured

ospf6_set_if_metric_value

This function sets the metric assigned to the given interface.

Syntax

```
int
ospf6_set_if_metric_value (char *tag, int ifindex,
                           int instid, int metric);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`metric`	Cost-metric value to be set: <1–65335>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_COST_INVALID when the cost-metric value is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_retrans_interval

This function sets the duration in seconds between LSA retransmissions for adjacencies belonging to the given interface.

Syntax

```
int
ospf6_set_if_retrans_interval (char *tag, int ifindex,
                               int instid, int interval);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`interval`	Value of interval between LSA retransmissions to be stored <1–65335>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_RETRANSMIT_INTERVAL_INVALID when the retransmit interval is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_rtr_dead_interval

This function sets the number of seconds that a router's Hello packets have not been seen on the given interface before its neighbors declare the router down.

Syntax

```
int
ospf6_set_if_rtr_dead_interval (char *tag, int ifindex,
                                int instid, int interval);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`interval`	Value of the router-dead interval to be stored (seconds): <1–65335>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_DEAD_INTERVAL_INVALID when the router-dead interval is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_rtr_priority

This function sets the priority value of the given interface.

Syntax

```
int
ospf6_set_if_rtr_priority (char *tag, int ifindex,
                           int instid, int priority);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`priority`	Interface priority value <0–255>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_PRIORITY_INVALID when the priority is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_transit_delay

This function sets the transit-delay value of the given OSPFv3 interface. This value is an estimate of the number of seconds required to transmit a link-state update packet through the interface.

Syntax

```
int
ospf6_set_if_transit_delay (char *tag, int ifindex,
                            int instid, int delay);
```

Input Parameters

`tag`	OSPFv3 process tag

`ifindex`	Interface index
`instid`	Instance identifier
`delay`	Transit-delay value in seconds <1–65335>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_IF_TRANSMIT_DELAY_INVALID when the transit delay is invalid

OSPF6_API_SET_ERR_IF_INSTANCE_ID_INVALID when the instance identifier is invalid

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

ospf6_set_if_type

This function sets the interface type of the given OSPFv3 interface.

Syntax

```
int
ospf6_set_if_type (char *tag, int ifindex, int instid, int type);
```

Input Parameters

`tag`	OSPFv3 process tag
`ifindex`	Interface index
`instid`	Instance identifier
`type`	Interface types:
`OSPF_IFTYPE_BROADCAST`	Broadcast
OSPF_IFTYPE_NBMA	Non-broadcast multi-access (NBMA)
`OSPF_IFTYPE_POINTOPOINT`	Point-to-Point
`OSPF_IFTYPE_POINTOMULTIPOINT`	Point-to-Multipoint

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_IF_NETWORK_TYPE_INVALID when the network configured on the interface is invalid

ospf6_set_reference_bandwidth

This function sets the reference bandwidth in Kbps. The reference value is used for calculating the default interface metric.

Syntax

```
int
ospf6_set_reference_bandwidth (char *tag, int refbw);
```

Input Parameters

`tag`	OSPFv3 process tag
`refbw`	Reference bandwidth value. The default value is 100,000 Kbps.

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

OSPF6_API_SET_ERR_REFERENCE_BANDWIDTH_INVALID when the bandwidth is not within the range

ospf6_set_restart_interval

This function sets the interval for the graceful restart timeout for the configured OSPFv3.

Syntax

```
int
ospf6_set_restart_interval (char *tag, int seconds);
```

Input Parameters

`tag`	OSPFv3 process tag
`seconds`	Interval for graceful restart timeout

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_GRACE_PERIOD_INVALID when an invalid grace period is set

OSPF6_API_SET_ERR_INVALID_REASON when the reason is invalid

ospf6_set_restart_support

This function sets the router's support for OSPF graceful restart.

Syntax

```
int
ospf6_set_restart_support(char *tag, int method);
```

Input Parameters

`tag`	OSPFv3 process tag
`method`	Status of support for OSPF graceful restart

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

ospf6_set_router_id

This function sets the router identifier, which uniquely identifies the router in the Autonomous System.

Syntax

```
int
ospf6_set_router_id (char *tag, struct pal_in4_addr addr);
```

Input Parameters

`tag`	OSPFv3 Process tag
`addr`	The valid 32-bit router identifier

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the OSPFv3 process does not exist

ospf6_set_stub_metric

This function sets the metric value advertised for the default route into Stub and NSSA areas.

Syntax

```
int
ospf6_set_stub_metric (char *tag, struct pal_in4_addr area_id, int metric);
```

Input Parameters

`tag`	OSPFv3 Process tag
`area_id`	Area identifier in an IPv4 address format
`metric`	The value of the cost metric to be set <0 –16777215>

Output Parameters

None

Return Values

OSPF6_API_SET_SUCCESS when the function succeeds

OSPF6_API_SET_ERROR when the function fails

OSPF6_API_SET_ERR_VR_NOT_EXIST when the virtual router does not exist

OSPF_API_SET_ERR_AREA_IS_BACKBONE when the given area is a backbone area

OSPF6_API_SET_ERR_METRIC_INVALID when the metric is invalid

OSPF6_API_SET_ERR_PROCESS_NOT_EXIST when the process does not exist

OSPF6_API_SET_ERR_AREA_NOT_EXIST when the area does not exist

OSPF6_API_SET_ERR_AREA_IS_DEFAULT when the area is default

Appendix A Message and TLV Types

Message Types

Message Type	Value
Route Request Message	0x0001
Route Message	0x0002
LSP Established Message	0x0003
LSP Delete message	0x0004
Notification Message	0x0005

TLV Types

TLV Type	Value
Re-Optimize TLV	0x0100
Retry TLV	0x0101
Hop Limit TLV	0x0102
Adm. Control TLV	0x0103
Priority TLV	0x0104
Path TLV	0x0105
ERO TLV	0x0106
LSP TLV	0x0107
Status TLV	0x0108
TE Class TLV	0x0109
Exclude Path TLV	0x0110
Extended Status TLV	0x0111

Protocol Types

Protocol Type	Value
RSVP-TE	0x00
CR-LDP	0x01

Status Codes

Status Code	Value
CSPF_CODE_SHUTDOWN	0x4000
CSPF_CODE_ROUTE_NOT_FOUND	0x4001
CSPF_CODE_ROUTE_FAIL	0x4002

Appendix B Source Code

This appendix provides an overview of OSPFv2 and OSPFv3 source code.

Source Code Relationships

OSPFv2

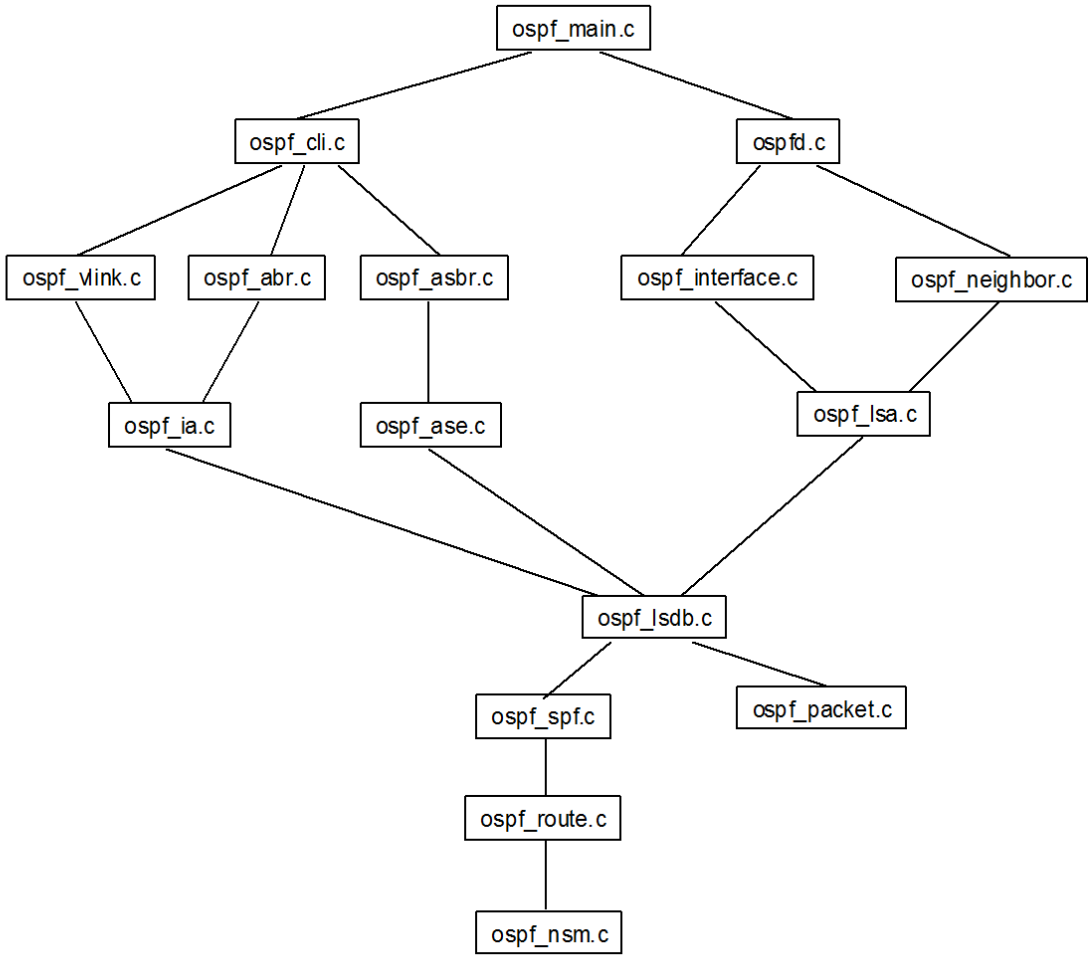


Figure B-1: OSPFv2 Source Code Relationships

OSPFv3

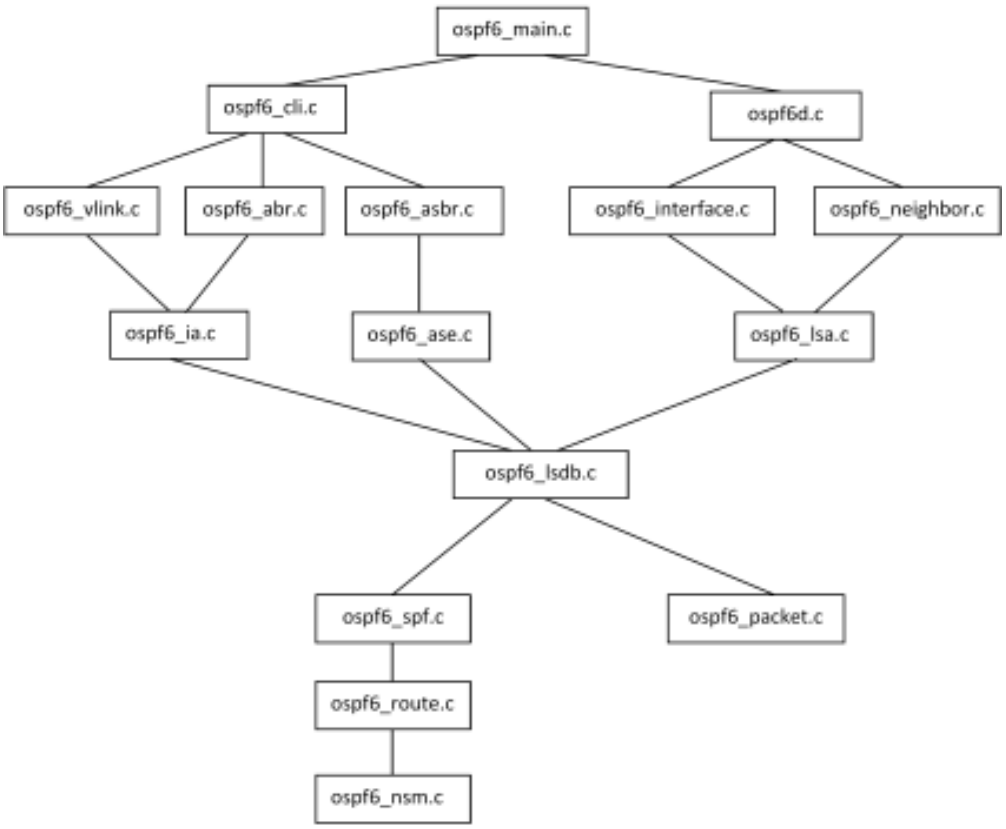


Figure B-2: OSPFv3 Source Code Relationships

Core Modules

OSPFv2

- The code for OSPF supporting IPv4
- Relative path: \ZebOS5\ospfd

Module	Description
ospf_main.c	Contains main routine to start OSPF
ospfd.c	Contains OSPF main functions and OSPF instance handling routines
ospf_vlink.c	OSPF virtual link handling routines
ospf_abr.c	OSPF router's behavior as area border router
ospf_asbr.c	OSPF router's behavior as autonomous system boundary router
ospf_interface.c	Contains OSPF's logical interface handling functions

Module	Description
ospf_ifsm.c	Contains OSPF's interface state machine
ospf_neighbor.c	Contains OSPF's neighbor handling functions
ospf_nfsm.c	Contains OSPF's neighbor state machine
ospf_ia.c	Contains routines for handling OSPF's inter-area routes
ospf_ase.c	Contains routines for handling OSPF's AS-External routes
ospf_lsa.c	Contains OSPF's Link State Advertisement (LSA) handling functions
ospf_lsdb.c	OSPF's link state database handling functions
ospf_spf.c	Contains routines that carry out SPF calculation
ospf_route.c	Handles manipulation and updating of routes; notifies NSM
ospf_nsm.c	Message processing for NSM and external routing information handling
ospf_api.c	Prototypes, functions and data structures for the CLI, MIB and other APIs
ospf_bfd_api.c	Contains routines and data structures for the OSPF and BFD API
ospf_bfd.c	Contains the routines for handling OSPF's BFD
ospf_bfc_cli.c	Contains the definitions for the BFD CLI commands
ospf_bfd_show.c	Contains routines to display OSPF BFD routing processes
ospf_cli.c	Contains the definitions for the OSPF CLI APIs
ospf_debug.c	Contain routines and data structures for debugging ospfd
ospf_flood.c	Provides for the flooding services of LSAs, depending on the particular role of the router at the time LSAs are either received or self-generated
ospf_mpls.c	Contains routines for OSPF MPLS processing
ospf_multi_area_link.c	Contains routines for multiple OSPF area on the IP Interface
ospf_network.c	Contains routines for OSPF networks processing
ospf_packet.c	Contains routines for OSPF packet processing
ospf_restart.c	Functions and data structures to restart
ospf_routemap.c	Functions and data structures to set route map attributes
ospf_show.c	Contains routines to display OSPF routing processes
ospf_snmp.c	Functions and data structures for handling MIB requests
ospf_te.c	Contains the routines for handling the OSPF Traffic Engineering
ospf_vrf.c	Contains the routines for handling the OSPF VRF

OSPFv3

The code for OSPF supporting IPv6

Relative path: \ZebOS5\ospf6d

Module	Description
ospf6_main.c	Contains main routine to start OSPFv3
ospf6d.c	Contains routines for handling OSPFv3 instance
ospf6_vlink.c	OSPFv3 virtual link handling routines
ospf6_abr.c	OSPFv3 router's behavior as area border router
ospf6_asbr.c	OSPFv3 router's behavior as autonomous system boundary router
ospf6_area.c	Contains the routines for handling OSPF's area
ospf6_interface.c	Contains OSPFv3's logical interface handling functions
ospf6_ifsm.c	Contains OSPFv3's interface state machine
ospf6_neighbor.c	Contains OSPFv3's neighbor handling functions
ospf6_nfsm.c	Contains OSPFv3's neighbor state machine
ospf6_ia.c	Contains routines for handling OSPFv3's inter-area routes
ospf6_ase.c	Contains routines for handling OSPFv3's AS-External routes
ospf6_lsa.c	Contains OSPFv3's Link State Advertisement (LSA) handling functions
ospf6_lsdb.c	OSPFv3's link state database handling functions
ospf6_spf.c	Contains routines that carry out SPF calculation
ospf6_route.c	Handles manipulation and updating of routes; notifies NSM
ospf6_nsm.c	Message processing for NSM and external routing information handling
ospf6_api.c	Prototypes, functions and data structures for the CLI, MIB and other APIs
ospf6_bfd_api.c	Function and data structures for the OSPFv3 and BFD API
ospf6_bfd.c	Contains the routines for handling OSPF's BFD
ospf6_bfc_cli.c	Contains the definitions for the OSPFv3 BFD CLI commands
ospf6_bfd_show.c	Contains routines to display OSPFv3 BFD routing processes
ospf6_cli.c	Contains the definitions for the OSPFv3 CLI commands
ospf6_debug.c	Contain routine for handling OSPFv3 debugging
ospf6_flood.c	Provides for the flooding services of LSAs, depending on the particular role of the router at the time LSAs are either received or self-generated
ospf6_network.c	Contains routines for OSPFv3 networks processing
ospf6_restart.c	Functions and data structures to restart
ospf6_routemap.c	Contains routines and data structures to set route map attributes
ospf6_show.c	Contains routines to display OSPFv3 routing processes
ospf6_snmp.c	Contains routines and data structures for handling MIB requests
ospf6_te.c	Contains routines for handling OSPFv3 Traffic Engineering

Module	Description
ospf6_vrf.c	Contains routines for OSPFv3 VRF
ospf6_packet.c	Contains routines for OSPFv3 packet processing

Index

A

C

G

H

I

L

M

O

P

R

S

T

V