ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Common Data Structures
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

Each ZebOS-XP developer guide describes data structures that are unique to that protocol or module. This guide describes data structures that are used by multiple ZebOS-XP protocols or modules.

Audience

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

Conventions

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

Table P-1: Conventions

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

Contents

This guide contains this chapter:

- Chapter 1, *Common Data Structures*

Related Documents

Use this guide along with the ZebOS-XP developer guide for the protocol or module you are extending or customizing.

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 Common Data Structures

This chapter describes the data structures that are used by multiple ZebOS-XP modules:

- bridge_id
- cli
- connected
- interface
- lib_globals
- nsm_bridge
- nsm_bridge_master
- nsm_master
- pal_in4_addr
- pal_in6_addr
- prefix
- prefix_ipv4
- prefix_ipv6
- rib
- stream
- thread
- variable

bridge_id

This data structure represents a bridge priority and bridge MAC address of a connected node. This data structure is defined in the `lib/L2/l2lib.h` file.

Member	Description
`prio`	Bridge priority
`addr`	Bridge MAC address

Definition

```
struct bridge_id
{
  unsigned char prio[2];
  unsigned char addr[ETHER_ADDR_LEN];
};
```

cli

This data structure represents a command that a user has given. This data structure is defined in the `lib/cli.h` file.

Member	Description
`cel`	CLI element
`str`	User input string
`out_func`	Output function used by `cli_out`
`out_val`	Output function's first argument
`line`	Arbitrary information for line
`auth`	Authorization required
`source`	Input source
`line_type`	For line
`min`	Line minimum
`max`	Line maximum
`index`	Real CLI
`index_sub`	Real CLI
`mode`	Real CLI
`status`	Current CLI status
`flags`	Flags
`self`	Arbitrary information for self
`privilege`	Privilege level
`ctree`	CLI tree
`zg`	Daemon-spefic library globals
`vr`	Global variable
`lines`	Terminal length
`callback`	Call back function
`cleanup`	Call back function
`show_func`	Call back function
`type`	Type of CLI

Member	Description
`count`	Total count
`current`	Arbitrary information about current node
`arg`	Look up argument
`afi`	Address family information
`safi`	Specific address family information
`port_range`	Layer 2 handling
`cv`	Vector used by IMI to encode a single command string

Definition

```
struct cli
{
  /* CLI element.  */
  struct cli_element *cel;

  /* User input string.  */
  char *str;

  /* Output function to be used by cli_out().  */
  CLI_OUT_FUNC out_func;

  /* Output function's first argument.  */
  void *out_val;

  /* Alternate storage for cli_out() message */
  char *out_snoop_buf;

  /* Arbitrary information for line.  */
  void *line;
  int min;
  int max;

  /* Auth required.  */
  int auth;

  /* Input source.  */
  int source;
#define CLI_SOURCE_USER                 0
#define CLI_SOURCE_FILE                 1

  /* For "line". */
  int line_type;
  int min;
  int max;

  /* Real CLI.  */
  void *index;
  void *index_sub;
  int mode;
/* Current CLI status.  */
  enum {
    CLI_NORMAL,
    CLI_CLOSE,
    CLI_MORE,
    CLI_CONTINUE,
    CLI_MORE_CONTINUE,
    CLI_WAIT
  } status;

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

  void *self;
  u_char privilege;
  struct cli_tree *ctree;

  /* Global variable.  */
  struct lib_globals *zg;
  struct ipi_vr *vr;

  /* Terminal length.  */
  int lines;

  /* Call back function.  */
  int (*callback) (struct cli *);
  int (*cleanup) (struct cli *);
  s_int32_t (*show_func) (struct cli *);
  int type;
  u_int32_t count;
  void *current;
  void *arg;
  afi_t afi;
  safi_t safi;

#ifdef HAVE_CUSTOM1
  /* L2 handling.  */
  u_int64_t port_range;
#endif /* HAVE_CUSTOM1 */

  /* Vector used by IMI to encode a single CLI command string.
   */
  cfg_vect_t *cv;
  /* Parse result saved for non-interactive shells */
  unsigned parse_result;
};
```

connected

This data structure represents all connected addresses. This data structure is defined in the `lib/if.h` file.

Member	Description
`next`	Pointer to the next connected network
`prev`	Pointer to the previous connected network
`ifp`	Attached interface
`family`	Address family for prefix
`conf`	Flags for configuration
`storageType`	Storage type
`flags`	Flags for connected address
`address`	Address of connected network
`destination`	Address of destination connected network
`label`	Label for Linux 2.2.X and upper
`lib_connected_cdr_ref`	Lib connected checkpoint database record reference
`ip_last_changed`	IP MIB timestamp
`ip_created`	IP MIB timestamps
`ip_enabled`	IPv4 capability is enabled

Definition

```
struct connected
{
  struct connected *next;
  struct connected *prev;

  /* Attached interface. */
  struct interface *ifp;

  /* Address family for prefix. */
  u_int8_t family;

  /* Flags for configuration. */
  u_int8_t conf;
#define NSM_IFC_REAL            (1 << 0)
#define NSM_IFC_CONFIGURED      (1 << 1)
#define NSM_IFC_ARBITER         (1 << 2)
#define NSM_IFC_ACTIVE           (1 << 3)
#define NSM_IFC_MARKED           (1 << 4)

  u_int8_t   storageType;
#define NSM_IFC_STORAGETYPE_OTHER    1
#define NSM_IFC_STORAGETYPE_VOLATILE 2
#define NSM_IFC_STORAGETYPE_NONVOLATILE 3
#define NSM_IFC_STORAGETYPE_PERMANENT 4
#define NSM_IFC_STORAGETYPE_READONLY 5

  /* Flags for connected address. */    /* XXX-VR */
  u_int8_t flags;
#define NSM_IFA_SECONDARY        (1 << 0)
#define NSM_IFA_ANYCAST          (1 << 1)
#define NSM_IFA_VIRTUAL          (1 << 2)
#ifdef HAVE_VRX
#define NSM_IFA_VRX_WRP          (1 << 3)
#endif /* HAVE_VRX */
#ifdef HAVE_HA
#define NSM_IFA_HA_DELETE        (1 << 4)
#endif /* HAVE_HA */
#ifdef HAVE_VRRP_V3
#define NSM_IFA_VRRP             (1 << 5)
#endif /* HAVE_VRRP_V3 */

  /* Address of connected network. */
  struct prefix *address;
  struct prefix *destination;

  /* Label for Linux 2.2.X and upper. XXX-VR */
  char *label;

#ifdef HAVE_HA
 HA_CDR_REF lib_connected_cdr_ref;
#endif /* HAVE_HA */

  /*IP MIB timestamps*/
  pal_time_t ip_last_changed;
  pal_time_t ip_created;

#ifdef HAVE_L3
  /* IPv4 capability is enabled */
  bool_t ip_enabled;
#endif /* HAVE_L3 */

};
```

interface

This data structure represents an interface. This data structure is defined in the `lib/if.h` file.

Member	Description
`name`	Interface name
`orig`	Interface name mapping
`ifindex`	Interface index
`cindex`	Interface attribute update flags
`flags`	Interface flags
`status`	ZebOS-XP internal interface status
`storm_control_properties`	Storm control featrues
`metric`	Interface metric
`mtu`	Interface MTU
`duplex`	Interface duplex status
`autonego`	Interface auto-negotiation
`flowcontrol`	Flow control
`autoneg_bits_advt`	Interface auto-negotiation bits (IANA notation) for advertisement
`autoneg_fault_advt`	Interface auto-negotiation fault for advertisement
`mdix`	Medium Dependent Interface with crossover
`arp_ageing_timeout`	Interface ARP aging timeout
`arp_p`	Neighbor discovery
`slot_id`	Slot identifier
`hw_type`	Hardware address (type)
`hw_addr`	Hardware address
`hw_addr_len`	Hardware address length
`bandwidth`	Interface bandwidth, bytes per second
`if_linktrap`	Interface link up or link down traps
`trap_linkup`	SNMP server link trap
`trap_linkdown`	SNMP server link trap

Member	Description
`if_alias`	Interface alias name
`conf_flags`	Whether the bandwidth has been configured or read from the kernel
`desc`	Description of the interface
`mau_default_type`	`ifMauDefaultType` object support (MAU MIB); should be assigned with a valu from enum `nsm_iana_if_mau_type_list_bits`
`default_duplex`	`ifMauDefaultType` object support (MAU MIB)
`default_bandwidth`	`ifMauDefaultType` object support (MAU MIB)
`ifc_ipv4`	Connected address list
`ifc_ipv6`	Connected IPv6 address list
`unnumbered_ipv4`	Unnumbered interface list
`unnumbered_ipv6`	Unnumbered IPv6 interface list
`info`	Daemon specific interface data pointer
`vr`	Pointer to virtual router context
`vrf`	Pointer to VRF context
`stats`	Statistics fields
`tunnel_if`	Tunnel interface
`ls_data`	Label space
`admin_group`	Administrative group to which this interface belongs to
`max_resv_bw`	Maximum amount of bandwidth that can be reserved (bytes)
`bw_constraint`	Bandwidth constraint per class type (bytes)
`tecl_priority_bw`	Available bandwidth at priority “P” (range 0 to 8)
`bind`	Bind information
`num_dl`	Number of data links. Based on this information, the system either uses the tree interface (datalink less than 1) or uses the pointer to the datalink structure. This is GMPLS information.
`gmpls_type`	Type, which may include unknown, data, control, and data-control
`gifindex`	GMPLS interface index
`phy_prop`	GMPLS interface common properties
`dlink`	Pointer to datalink
`port`	Port information

Member	Description
bridge_name	Bridge name
lacp_admin_key	LACP administration key
agg_param_update	LACP aggregator update flag
lacp_agg_key	LACP aggregator key
bc_mode	Bandwidth constrain mode for each interface
vrx_flag	Flag for virtual router for CheckPoint VSX type
local_flag	Local source
vrx_if_info	Related VRX information
ifLastChange	Time for last status change
pid	Process ID
type	Interface type L2/L3
config_duplex	Stores the configured duplex value
trust_state	QoS set trust state for port
vlan_classifier_group_id	VLAN classifier group ID
clean_pend_resp_list	List of NSM clients pending response on interface delete message
interface_cdr_ref	High Availability (HA) interface Checkpoint Abstraction Layer (CAL) created record reference value
chkpt_info	MPLS HA
nsm_mpls_if_cdr_ref	MPLS HA
rmap_if_match_cmd_list	Route map match interface set list
sync_params	LDP-IGP synchronization parameters
nsm_band_width_profile	Interface BW- Configured CIR/EIR sync
rmap_name	PBR: route map tag
rmap_type	PBR: route map type
rmap_status	PBR: route map status
ipv6_enabled	IPv6 capability is enabled
ipv6_forwarding	IPv6 capability is enabled
ip_enabled	IPv4 capability is enabled

Definition

```
struct interface
{
  /* Interface name. */
  char name[INTERFACE_NAMSIZ + 1];
#ifdef HAVE_INTERFACE_NAME_MAPPING
  char orig[INTERFACE_NAMSIZ + 1];
#endif /* HAVE_INTERFACE_NAME_MAPPING */

  /* Interface index. */
  s_int32_t ifindex;

  /* Interface attribute update flags.  */
  u_int32_t cindex;

  /* Interface flags. */
  u_int32_t flags;

  /* ZebOS internal interface status */
  u_int32_t status;

  /* Storm Control Features */
  struct storm_control storm_control_properties;

#define NSM_INTERFACE_ACTIVE            (1 << 0)
#define NSM_INTERFACE_ARBITER           (1 << 1)
#define NSM_INTERFACE_MAPPED            (1 << 2)
#define NSM_INTERFACE_MANAGE            (1 << 3)
#define NSM_INTERFACE_DELETE            (1 << 4)
#define NSM_INTERFACE_IPV4_UNNUMBERED   (1 << 5)
#define NSM_INTERFACE_IPV6_UNNUMBERED   (1 << 6)
#ifdef HAVE_HA
#define HA_IF_STALE_FLAG                (1 << 7)
#endif /* HAVE_HA */
#define IF_HIDDEN_FLAG                  (1 << 8)
#define IF_NON_CONFIGURABLE_FLAG        (1 << 9)
#define IF_NON_LEARNING_FLAG            (1 << 10)
/*in running config this should be display only after the phy Intf*/
#define IF_DEPENDS_ON_PHY_INTF          (1 << 11)
#define NSM_IF_NO_NOTIFICATION_IMI      (1 << 12)
#ifdef HAVE_MT
#define NSM_INTERFACE_NS_CHANGE         (1 << 13)
#endif /* HAVE_MT && ENABLE_PAL_PATH*/
#define IF_ARP_REFRESH_DISABLED         (1 << 15)
#ifdef HAVE_DHCP_CLIENT
#define IF_ADDRESS_DHCP                 (1 << 16)
#define IF_ADDRESS_DHCP6                (1 << 17)
#endif
```

```
  /* Interface metric */
  s_int32_t metric;

  /* Interface MTU. */
  s_int32_t mtu;

  /* Interface DUPLEX status. */
  u_int32_t duplex;

  /* Interface AUTONEGO. */
  u_int32_t autonego;

  /* Flowcontrol */
  u_int8_t flowcontrol;
#define FLOW_SEND_ON                    (1 << 0)
#define FLOW_RECEIVE_ON                 (1 << 1)

  /* Interface Auto-Negotiation bits (IANA notation) for advertisement */
  u_int32_t autoneg_bits_advt;

  /* Interface Auto-Negotiation fault for advertisement */
  u_int8_t autoneg_fault_advt;

  /* Interface MDIX crossover. */
  u_int32_t mdix;

  /* Interface ARP AGEING TIMEOUT. */
  u_int32_t arp_ageing_timeout;

#ifdef HAVE_NDD
  struct arp_params arp_p;
#endif /* HAVE_NDD */

  /* Slot Id. */
  u_int32_t slot_id;

  /* Hardware address. */
  u_int16_t hw_type;
  u_int8_t hw_addr[INTERFACE_HWADDR_MAX];

  s_int32_t hw_addr_len;

#define LIB_UNIT_KBPS            1000
  /* interface bandwidth, bytes/s */
  float64_t bandwidth;

  /*Interface link up or link down traps */
  s_int32_t if_linktrap;

  #ifdef HAVE_SNMP_AGENT
  /*snmp-server enable link trap*/
  s_int32_t trap_linkup;
  s_int32_t trap_linkdown;
  #endif /* HAVE_SNMP_AGENT */

  /* Interface alias name */
  char if_alias[INTERFACE_NAMSIZ + 1];

  /* Has the bandwidth been configured/read from kernel. */
  char conf_flags;
#define NSM_BANDWIDTH_CONFIGURED    (1 << 0)
#define NSM_MAX_RESV_BW_CONFIGURED  (1 << 1)
//#define NSM_SWITCH_CAP_CONFIGURED   (1 << 2)
#define NSM_DUPLEX_CONFIGURED       (1 << 2)
#define NSM_MIN_LSP_BW_CONFIGURED   (1 << 3)
#define NSM_MAX_LSP_SIZE_CONFIGURED (1 << 4)
  /* description of the interface. */
  char *desc;

#ifdef HAVE_L2
  /* ifMauDefaultType object support (MAU MIB). Should be assigned with a value
  * from enum nsm_iana_if_mau_type_list_bits */
  u_int8_t mau_default_type;
  u_int32_t default_duplex;
  float default_bandwidth;
#endif /* HAVE_L2 */

  /* Connected address list. */
  struct connected *ifc_ipv4;
#ifdef HAVE_IPV6
  struct connected *ifc_ipv6;
#endif /* HAVE_IPV6 */

  /* Unnumbered interface list.  */
  struct list *unnumbered_ipv4;
#ifdef HAVE_IPV6
  struct list *unnumbered_ipv6;
#endif /* HAVE_IPV6 */

  /* Daemon specific interface data pointer. */
  void *info;

  /* Pointer to VR/VRF context. */
  struct ipi_vr *vr;
  struct ipi_vrf *vrf;

  /* Statistics fileds. */
  struct pal_if_stats stats;

#ifdef HAVE_TUNNEL
```

```
  /* Tunnel interface. */
  struct tunnel_if *tunnel_if;
#endif /* HAVE_TUNNEL */

#ifdef HAVE_MPLS
  /* Label space */
  struct label_space_data ls_data;
#endif /* HAVE_MPLS */

#ifdef HAVE_TE
  /* Administrative group that this if belongs to */
  u_int32_t admin_group;

  /* Maximum reservable bandwidth (bytes/s) */
  float32_t max_resv_bw;

#ifdef HAVE_DSTE
  /* Bandwidth constraint per class types (bytes/s) */
  float32_t bw_constraint[MAX_BW_CONST];
#endif /* HAVE_DSTE */

  /* Available bandwidth at priority p, 0 <= p < 8 */
  float32_t tecl_priority_bw [MAX_PRIORITIES];

#endif /* HAVE_TE */

  /* Bind information.  */
  u_char bind;
#define NSM_IF_BIND_VRF           (1 << 0)
#define NSM_IF_BIND_MPLS_VC       (1 << 1)
#define NSM_IF_BIND_MPLS_VC_VLAN  (1 << 2)
#define NSM_IF_BIND_VPLS          (1 << 3)
#define NSM_IF_BIND_VPLS_VLAN     (1 << 4)

#ifdef HAVE_GMPLS
  /* GMPLS information */
  /* Number of data links. Based on this information we will either use the
    tree if dl > 1 or use the pointer to the datalink structure */
  u_char num_dl;

  /* Type includes unknow/data/control/data-control */
  u_char gmpls_type;

  u_int32_t gifindex;

  /* GMPLS interface common properties */
  struct phy_properties phy_prop;

  /* Pointer to datalink */
  union
```

```
    {
      struct avl_tree *dltree;
      struct datalink *datalink;
    }dlink;

#endif /* HAVE_GMPLS */

#ifdef HAVE_L2
  void *port;
  char bridge_name[INTERFACE_NAMSIZ + 1 ];
#endif /* HAVE_L2 */

#ifdef HAVE_LACPD
  u_int16_t lacp_admin_key;
  u_int16_t agg_param_update;
  u_int32_t lacp_agg_key;
#endif /* HAVE_LACPD */

#ifdef HAVE_DSTE
  /* Bandwdith constrain mode for every interface. */
  bc_mode_t bc_mode;
#endif /* HAVE_DSTE */

#ifdef HAVE_VRX
  u_char vrx_flag;
#define IF_VRX_FLAG_NORMAL             0
#define IF_VRX_FLAG_WRPJ               1
#define IF_VRX_FLAG_WRP                2

  /* Local src. */
  u_char local_flag;
#define IF_VRX_FLAG_LOCALSRC           1

  /* Related VRX information. */
  struct vrx_if_info *vrxif;
#endif /* HAVE_VRX */
  pal_time_t ifLastChange;

#ifdef HAVE_CUSTOM1
  int pid;
#endif /* HAVE_CUSTOM1 */
  u_char type; /* Interface type L2/L3 */
#define IF_TYPE_L3 0
#define IF_TYPE_L2 1

#ifdef HAVE_TDM_VC
#define IF_TYPE_TDM 2
#endif

/* Maximum L2 MTUs */
```

```
#if defined (HAVE_VLAN_STACK) || defined (HAVE_PROVIDER_BRIDGE)
  #define IF_ETHER_L2_DEFAULT_MTU 1526
#elif defined (HAVE_VLAN)
  #define IF_ETHER_L2_DEFAULT_MTU 1522
#else
  #define IF_ETHER_L2_DEFAULT_MTU 1518
#endif

  /* To store the configured duplex value */
  u_char config_duplex;

#ifdef HAVE_QOS
  int trust_state;
#endif /* HAVE_QOS */

#ifdef HAVE_VLAN_CLASS
 u_int32_t vlan_classifier_group_id;
#endif /* HAVE_VLAN_CLASS */

 struct list *clean_pend_resp_list;

#ifdef HAVE_HA
 HA_CDR_REF interface_cdr_ref;
#ifdef HAVE_MPLS
 s_int32_t chkpt_info;
 HA_CDR_REF nsm_mpls_if_cdr_ref;
#endif /* HAVE_MPLS */
#endif /* HAVE_HA */

  struct list *rmap_if_match_cmd_list;

  /*LDP-IGP Sync */
  void *sync_params;

  /*Interface BW- Configured CIR/EIR sync*/
  struct nsm_band_width_profile *bw_profile;
#ifdef HAVE_PBR
#define NSM_NO_RMAP      0
#define NSM_IP_RMAP      1
#define NSM_IPV6_RMAP    2
   char *rmap_name;
   u_int8_t rmap_type; /*ip/ipv6*/
#define NSM_RMAP_INACTIVE  0
#define NSM_RMAP_ACTIVE    1
   u_int8_t rmap_status;
#endif /*HAVE_PBR*/

#ifdef HAVE_IPV6
  /* IPv6 capability is enabled */
  bool_t ipv6_enabled;
  bool_t ipv6_forwarding;
#endif /* HAVE_IPV6 */

#ifdef HAVE_L3
  /* IPv4 capability is enabled */
  bool_t ip_enabled;
#endif /* HAVE_L3 */
};
```

lib_globals

This data structure represents daemon-specific library globals. Each instance of a daemon maintains an instance of this structure. This data structure is defined in the `lib/lib.h` file.

Member	**Description**
`progname`	Daemon name
`hostname`	Host name
`protocol`	Module ID defined in `pal/api/pal_modules.def`
`flags`	Flags
`stop_cause`	Module stop callbacks
`motd`	Banner configuration
`logging_level_list`	Show logging level
`module_severity`	Module (process) severity
`cwd`	Current working directory
`master`	Thread master
`pend_read_thread`	Pending read threads
`ifg`	Interface masters
`vr_vec`	Virtual router vector
`vr_deleted_vec`	Virtual router vector
`fib2vrf`	VRF vector for Kernel Table ID mapping
`log`	Logging information
`log_default`	Default log
`disable_log`	HAdisable log
`snmp`	SNMP: agentx or SMUX
`gifindex`	GMPS interface index
`vr_callback`	Callback function
`vrf_callback`	Callback function
`user_callback`	Callback function
`host_user`	Callback function

Member	Description
`nc`	NSM client
`rc`	ribd client
`mcast`	Multicast globals
`bs`	BFD server
`bc`	BFD client
`vlog_clt`	VLOG client
`vlog_file_set_cb`	Callback installed by both IMI and VLOGD
`vlog_file_unset_cb`	Callback installed by both IMI and VLOGD
`vlog_file_get_cb`	Callback installed by both IMI and VLOGD
`os`	ONM server
`oc`	ONM client
`ms`	MSTP server
`mc`	MSTP client
`vs`	VPORT server
`vc`	VPORT client
`ping_flag`	Ping flag
`ping_fib`	Ping FIB
`vr_instance`	Virtual router instance
`pal_debug`	PAL debug
`pal_kernel`	PAL kernel
`pal_log`	PAL log
`pal_np`	PAL
`pal_np`	PAL
`pal_socket`	PAL socket
`pal_stdlib`	PAL standard library
`pal_string`	PAL string
`pal_time`	PAL time
`vty_master`	VTY master structure

Member	Description
`ctree`	CLI tree
`imh`	IMI message handler
`ss`	Show server
`alarm_smi_server`	SMI server alarm
`config_smi_server`	SMI server configuraiton
`pal_vrrp`	VRRPv2 PAL data structure pointer
`handle`	License manager handle
`t_check_expiration`	Timer thread to check the license expiration
`lib_ha`	High Availability structure
`am_ptr`	Availability Management pointer
`lib_fm`	Fault management - fault recording
`proto`	Protocol globals
`vr_in_cxt`	Virtual router currently in context
`vr_global`	Global virtual router
`ssock_cb_zombie_list`	Stream socket-CB zombies list
`cqueue_buf_free_list`	Circular queue buffers free list
`commsg`	Instance of COMMSG transport for this daemon
`lib_acl_ntf_cb`	Access-list add/delete notification callback
`gmif`	GMPLS interface should be allocated when needed
`mt_cxt`	Multitenancy context

Definition

```
struct lib_globals
{
  char progname[LIB_MAX_PROG_NAME+1];

  /* for hostname */
  char hostname[LIB_MAX_HOSTNAME+1];

  /* Module ID defined in pal/api/pal_modules.def.  */
  module_id_t protocol;
  /* Flags */
  u_int8_t flags;
#define LIB_FLAG_SHUTDOWN_IN_PROGRESS           (1 << 0)
#if defined HAVE_RESTART || defined HAVE_RSVP_GRST
#define LIB_FLAG_GRACEFUL_SHUTDOWN_IN_PROGRESS  (1 << 1)
#endif /* HAVE_RESTART || HAVE_RSVP_GRST */
  mod_stop_cause_t stop_cause;

  /* Banner configuration.  */
  char *motd;

 /* for show logging level */
 struct list *logging_level_list;

 /* module(process) severity */
 s_int32_t module_severity;

  /* Current working directory.  */
  char *cwd;

  /* Thread master. */
  struct thread_master *master;
  struct thread *pend_read_thread;

  /* Interface Master. */
  struct if_master ifg;

  /* VR vector. */
  vector vr_vec;

  /* VR vector. */
  vector vr_deleted_vec;

  /* VRF vector for Kernel Table ID mapping. */
  vector fib2vrf;
 /* Host */
  struct host *host;
  /* Log. */
  struct zlog *log;
  struct zlog *log_default;

#ifdef HAVE_HA
  bool_t disable_log;
#endif /* HAVE_HA */

#ifdef HAVE_SNMP
  /* snmp : agentx or smux */
  struct snmp_master snmp;
#endif /* HAVE_SNMP */
```

```
#ifdef HAVE_GMPLS
  s_int32_t gifindex;
#endif /* HAVE_GMPLS */

  /* Callback functions. */
  int (*vr_callback[VR_CALLBACK_MAX]) (struct ipi_vr *);
  int (*vrf_callback[VRF_CALLBACK_MAX]) (struct ipi_vrf *);
  int (*user_callback[USER_CALLBACK_MAX]) (struct ipi_vr *,
                                            struct host_user *);

  /* NSM client. */
  struct nsm_client *nc;

#ifdef HAVE_RIBD
  struct rib_client *rc;
#endif /* HAVE_RIBD */

#ifdef HAVE_MCAST
  struct mcast_globals *mcast;
#endif /* HAVE_MCAST */

#if defined (HAVE_BFD) || defined (HAVE_MPLS_OAM)
  /* BFD server.  This will be moved to bfd_globals later.  */
  struct bfd_server *bs;

  /* BFD client.  */
  struct bfd_client *bc;
#endif /* defined (HAVE_BFD) || defined (HAVE_MPLS_OAM) */

#ifdef HAVE_VLOGD
  /* VLOG client. */
  struct vlog_client *vlog_clt;

  /* Callbacks to be installed by IMI and VLOGD. */
  VLOG_SET_LOG_FILE_CB   vlog_file_set_cb;
  VLOG_UNSET_LOG_FILE_CB vlog_file_unset_cb;
  VLOG_GET_LOG_FILE_CB   vlog_file_get_cb;
#endif /* HAVE_VLOGD */

  /* ONM client. */
  struct onm_server *os;
  struct onm_client *oc;

  struct mstp_server *ms;
  struct mstp_client *mc;

#ifdef HAVE_VPORT
  struct vport_server *vs;
  struct vport_client *vc;
#endif /* HAVE_VPORT */
```

```
  /* VR. */

#ifdef HAVE_VRF
#ifdef HAVE_VRF_NS
  bool_t ping_flag;
  fib_id_t ping_fib;
#endif /*HAVE_VRF_NS*/
#endif /*HAVE_VRF*/
  u_int32_t vr_instance;

  /* PAL. */
  pal_handle_t pal_debug;
  pal_handle_t pal_kernel;
  pal_handle_t pal_log;
  pal_handle_t pal_np;
  pal_handle_t pal_socket;
  pal_handle_t pal_stdlib;
  pal_handle_t pal_string;
  pal_handle_t pal_time;

  /* Vty master structure. */
  struct vty_server *vty_master;

  /* CLI tree. */
  struct cli_tree *ctree;

  /* IMI message handler.  */
  struct message_handler *imh;

  /* Show server.  */
  struct show_server *ss;

#ifdef HAVE_SMI
  struct smi_server *alarm_smi_server;
  struct smi_server *config_smi_server;
#endif /*HAVE_SMI */

#ifdef HAVE_VRRPD
  /* VRRPv2 PAL data structure pointer. */
  pal_handle_t pal_vrrp;
#endif /* HAVE_VRRPD */

#ifdef HAVE_LICENSE_MGR
  lic_mgr_handle_t handle;
  struct thread *t_check_expiration;
#endif /* HAVE_LICENSE_MGR */

#ifdef HAVE_HA
  LIB_HA  lib_ha;
  AM_PTR  am_ptr;
```

```
#endif /* HAVE_HA */

  /* Fault Management - Fault Recording. */
  void *lib_fm;

  /* Protocol Globals. */
  void *proto;

  /* VR currently in context */
  struct ipi_vr *vr_in_cxt;

  /* Stream Socket-CB Zombies List */
  struct list *ssock_cb_zombie_list;

  /* Circular Queue Buffers Free List */
  struct cqueue_buf_list *cqueue_buf_free_list;

  /* Instance of COMMSG transport for this daemon. */
  COMMSG *commsg;

  /* Access-list add/delete notification callback. */
  filter_ntf_cb_t lib_acl_ntf_cb;

#ifdef HAVE_GMPLS
  /* gmpls_if should be allocated when needed */
  struct gmpls_if *gmif;
#endif /* HAVE_GMPLS */

#ifdef HAVE_MT
  /* MT currently in context */
  struct imish_mt *mt_cxt;
#endif /* HAVE_MT */
};
```

nsm_bridge

This data structure represents a NSM bridge (switch). This data structure is defined in the `nsm/L2/nsm_bridge.h` file.

Member	Description
`next`	nsm_bridge Next node
`pprev`	nsm_bridge previous node
`name`	Name
`type`	Type
`is_default`	Default bridge
`ageing_time`	Aging Time for FDB entries
`enable`	Spanning tree enable
`provider_edge`	Provider edge enable
`backbone_edge`	Back none edge enable
`bridge_id`	Bridge Identifier
`bridge_mac`	Bridge mac
`pbb_te_group_tree`	PBB TE group avl tree
`vip_port_map`	VIP port map
`learning`	Indicates whether the bridge is learning bridge
`avl_tree port_tree`	Port list
`port_id_mgr`	SNPM port ID manager
`static_fdb_list`	Static FDB list
`bridge_listener_list`	Bridge listener list
`vlan_table`	VLAN table
`stp_decoupled_vlan_tabl`	VLAN table
`svlan_table`	SVLAN table
`stp_decoupled_svlan_table`	Decoupled SVLAN table
`pro_edge_swctx_table`	Pro edge Switching Context table
`cvlan_reg_tab_list`	CVLAN list

Member	Description
vlan_listener_list	VLAN listener list
bvlan_table	bridged VLAN list
stp_decoupled_bvlan_table	Stp bridged VLAN decoupled list
event	Thread event
master	Back pointer to bridge master
gvrp	GVRP structure for the bridge
gmrp_list	GMRP structure for the bridge
gmrp_bridge	GMRP bridge
gmrp_ext_filter:1	Enable Extended filtering option
vlan_num_deletes	VLAN num deletes
br_conf	Pointer to configuration store
traffic_class_enabled	L2 traffic class status
topology_type	Topology type - none/ring - currently used to support RSTP
num_cosq	Number of Class of Service queues
eps_tree	eps tree
raps_tree	raps tree
dcbg	nsm dcb bridge
evc_tree	evc avl tree
routed_vlan_tree	AVL tree for Routed VLANs
max_mst_instances	Maximum number of mst instances
uni_type_mode	UNI type; enabled, disabled or not the bridge

Definition

```
struct nsm_bridge
{
  struct nsm_bridge *next;
  struct nsm_bridge **pprev;

  /* Name. */
  char name[NSM_BRIDGE_NAMSIZ + 1];

  /* Type. */
  u_int8_t type;

  u_int8_t is_default;

  /* Ageing Time for fdb entries */
  u_int32_t ageing_time;

  /* Spanning tree Enable */
  u_int16_t enable;

#ifdef HAVE_PROVIDER_BRIDGE
  u_int8_t provider_edge:1;
#endif /* HAVE_PROVIDER_BRIDGE */
#if defined (HAVE_I_BEB) || defined (HAVE_B_BEB)
  u_int8_t      backbone_edge:1;
  u_int32_t     bridge_id;
  u_int8_t      bridge_mac[ETHER_ADDR_LEN];
#endif /* HAVE_I_BEB || HAVE_B_BEB */

#if defined (HAVE_PBB_TE)
  struct avl_tree *pbb_te_group_tree;
#endif /* HAVE_I_BEB && HAVE_B_BEB && HAVE_PBB_TE */

#if defined (HAVE_I_BEB)
  u_int8_t      vip_port_map[512];
#endif /* HAVE_I_BEB */

#define NSM_BRIDGE_AGEING_DEFAULT    300
  /* learning. Whether the bridge is learning bridge or not */
  int learning;
#define NSM_LEARNING_BRIDGE_SET      1
#define NSM_LEARNING_BRIDGE_UNSET    0

  /* Port list. */
  struct avl_tree *port_tree;

#ifdef HAVE_SNMP
  struct bitmap *port_id_mgr;
#endif /* HAVE_SNMP */

  /* Static FDB List */

  struct ptree *static_fdb_list;

  struct list *bridge_listener_list;
#ifdef HAVE_VLAN
  /* VLAN table. */
  struct avl_tree *vlan_table;
  struct avl_tree *stp_decoupled_vlan_table;
  struct avl_tree *svlan_table;
  struct avl_tree *stp_decoupled_svlan_table;

  struct avl_tree *pro_edge_swctx_table;
  struct list *cvlan_reg_tab_list;
  struct list *vlan_listener_list;
#ifdef HAVE_B_BEB
  struct avl_tree *bvlan_table;
  struct avl_tree *stp_decoupled_bvlan_table;
#endif /* HAVE_B_BEB */
#endif /* HAVE_VLAN */

  /* Thread event. */
  struct thread *event;

  /* Back pointer to bridge master. */
  struct nsm_bridge_master *master;

#ifdef HAVE_GVRP
  /* Gvrp structure for the bridge */
  struct gvrp *gvrp;
#endif /* HAVE_GMRP */

#ifdef HAVE_GMRP
  /* Gmrp structure for the bridge */
  struct avl_tree *gmrp_list;
  struct gmrp_bridge *gmrp_bridge;
  unsigned char gmrp_ext_filter:1;
#endif /* HAVE_GVRP */

  u_int32_t vlan_num_deletes ;
 /* Pointer to configuration store */
  struct nsm_bridge_config *br_conf;
  u_int8_t traffic_class_enabled;
  /* Topology type - none/ring - currently used to support RRSTP */
  enum nsm_topology topology_type;
#ifdef HAVE_HA
  HA_CDR_REF nsm_bridge_cdr_ref;
#endif /* HAVE_HA */

#ifdef HAVE_QOS
  /* Num of Class of service queues */
  s_int32_t num_cosq;
#endif /* HAVE_QOS */

#ifdef HAVE_G8031
  struct avl_tree * eps_tree;
#endif /* HAVE_G8031 */

#ifdef HAVE_G8032
```

```
    struct avl_tree *raps_tree;
#endif  /*HAVE_G8032*/

#ifdef HAVE_DCB
   struct nsm_dcb_bridge *dcbg;
#endif /* HAVE_DCB */

#ifdef HAVE_EVC
   struct avl_tree *evc_tree;
#endif /* HAVE_EVC */

 /* AVL tree for Routed Vlans */
 struct avl_tree *routed_vlan_tree;

 u_int16_t max_mst_instances;

 u_int8_t uni_type_mode; ///< UNI Type enabled or disabled or not the bridge
};
```

nsm_bridge_master

This data structure represents the NSM bridge master. This data structure is defined in the `nsm/L2/nsm_bridge.h` file.

Member	Description
`bridge_list`	Bridge list
`flags`	Flags
`beb`	Backbone edge bridge
`nsm_instance_bmp`	NSM instance bitmap
`instance_map`	Instance map
`b_bridge`	Backbone bridge
`nm`	Back pointer to nsm_master
`group_tree`	AVL group tree
`rule_tree`	AVL rule tree
`event`	Thread event

Definition

```
struct nsm_bridge_master
{
  struct nsm_bridge *bridge_list;

  u_char flags;
#define CUSTOM_B_BEB                        (1 << 0)
#define CUSTOM_I_BEB                        (1 << 1)

#if defined(HAVE_I_BEB) || defined(HAVE_B_BEB)
  struct nsm_beb_bridge *beb;
#endif
#if defined HAVE_PBB_TE || defined HAVE_G8031 || \
   defined HAVE_G8032 || defined HAVE_G8032V2
   struct nsm_instance_bmp instanceBmp;
#endif /*HAVE_PBB_TE || defined HAVE_G8031 || defined HAVE_G8032 */

#if defined HAVE_PBB_TE || defined HAVE_G8031 || defined HAVE_G8032
  u_int8_t instance_map[NSM_INSTANCE_MAP_LEN];
#endif /*HAVE_PBB_TE*/

#ifdef HAVE_B_BEB
```

```
  struct nsm_bridge *b_bridge;
#endif

  /* Back pointer to nsm_master. */
  struct nsm_master *nm;

#ifdef HAVE_VLAN_CLASS
  struct avl_tree *group_tree;
  struct avl_tree *rule_tree;
#endif /* HAVE_VLAN_CLASS */

  /* Thread event. */
  struct thread *event;
};
```

nsm_master

This data structure contains NSM system-wide settings. This data structure is defined in `nsm/nsmd.h` file.

Member	Description
`vr`	Pointer to the virtual router
`zg`	NSM pointer to the library globals
`desc`	Description
`module_bits`	Control virtual router support per PM
`start_time`	NSM master start time
`multipath_num`	Maximum path configuration
`mp_size`	Maximum ECMP path limit
`max_static_routes`	Maximum static routes
`max_fib_routes`	Maximum FIB routes excluding kernel, connect and static
`max_frame_size`	Maximum frame size permissible
`fib_retain_time`	Time of retaining stale FIB routes when NSM start
`t_sweep`	Sweep stale FIB routes
`t_rib_kernel_sync`	RIB kernel sync
`label_pool_table`	Table of label pool managers
`label_pool_table`	Label pool table
`nmpls`	NSM MPLS top structure
`resource_counter`	QoS resource ID counter
`admin_group_array`	Admin group array
`rtadv`	Router Advertisement structure
`vrrp`	VRRP global data
`t_kernel_msg_stagger`	Hold timer to stagger writes to the kernel
`kernel_msg_stagger_list`	List for storing messages that need to be sent to the kernel
`bridge`	New bridge master
`l2_oam_master`	L2 OAM master
`nsm_layer2_master l2`	Layer 2 related information

Member	Description
`vmap`	VLAN access master
`class_rule_type`	VLAN classifier type global
`if_params`	Interface parameter list
`lacp_admin_key_mgr`	LACP administrator key manager
`ake_list`	LACP administrator key element
`psc`	Port selection criteria
`phyEntityList`	Physical Entity list
`last_change_time`	Disconnect time
`qos_state`	QoS state per virtual router
`nsm_qos_acl_master`	MAC and IP acl list
`acl`	MAC and IP ACL list
`cmap`	CMAP list
`cmap`	PMAP list
`mac_acl`	MAC access list master
`arp`	NSM arp master
`ipsec_master`	IPSEC NSM master
`firewall_master`	Firewall master
`nsm_master_cdr_ref`	NSM master Checkpoint Abstraction Layer (CAL) created record reference
`nsm_master_rib_sweep_tmr_cdr_ref`	NSM master rib sweep timer for Checkpoint Abstraction Layer (CAL) created record reference
`access_group_if`	List of access-list interface structure which keeps track of acl names configured on interfaces
`nmd_conf`	NSM configured and terminal debug flags
`nmd_term`	NSM configured and terminal debug flags
`route_map_if`	List of route map interface structure that keeps track of route maps configured on interfaces
`apbf_invalid_nh`	IPV4 invalid nexthop list
`apbf_invalid_nh6`	IPV6 invalid nexthop list
`nsm_apbf_nh4_tbr`	To Be Resolved list

Member	Description
nsm_apbf_nh4_res	Resolved list
nsm_apbf_nh4_nr	Not Resolved list
nsm_apbf_nh6_tbr	To Be Resolved list
nsm_apbf_nh6_res	Resolved list
nsm_apbf_nh6_nr	Not Resolved list

Definition

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

  /* NSM pointer to lib_globals */
  struct lib_globals *zg;
#define NSM_ZG   (nzg)

  /* Description. */
  char *desc;

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

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

  u_char flags;
#define NSM_MULTIPATH_REFRESH           (1 << 0)
#define NSM_FIB_RETAIN_RESTART          (1 << 1)
#define NSM_IPV4_FORWARDING             (1 << 2)
#define NSM_IPV6_FORWARDING             (1 << 3)

  /* Maximum path config. */
  u_char multipath_num;

  /* Maximum static routes */
  u_int32_t max_static_routes;

  /* Maximum FIB routes excluding Kernel, Connect and Static*/
  u_int32_t max_fib_routes;

 /* Maximum Frame Size permissible */
  u_int32_t max_frame_size;

  /* The time of retaining stale FIB routes when NSM start. */
  u_int16_t fib_retain_time;
#define NSM_FIB_RETAIN_TIME_MIN         1
#define NSM_FIB_RETAIN_TIME_MAX         65535
#define NSM_FIB_RETAIN_TIME_DEFAULT     60
#define NSM_FIB_RETAIN_TIME_FOREVER     0

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

  /* The following is the table of label pool managers that are
     handled by ZebOS-XP */
#if defined HAVE_GMPLS
  struct route_table *label_pool_table[GMPLS_LABEL_TYPE_MAX];
#elif defined HAVE_MPLS
  struct route_table *label_pool_table[1];
#endif /* HAVE_GMPLS */

#ifdef HAVE_MPLS
  /* NSM MPLS top structure. */
  struct nsm_mpls *nmpls;

#define NSM_MPLS   (nm->nmpls)
#endif /* HAVE_MPLS */

#ifdef HAVE_TE
  /* QOS resource id counter */
  u_int32_t resource_counter;

  struct admin_group admin_group_array [ADMIN_GROUP_MAX];
#endif /* HAVE_TE */

#ifdef HAVE_RTADV
  struct rtadv *rtadv;
#endif /* HAVE_RTADV */

#ifdef HAVE_VRRP
  struct vrrp_global *vrrp;
#endif /* HAVE_VRRP */

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

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

```
#ifdef HAVE_L2
  struct nsm_bridge_master *bridge;
#endif /* HAVE_L2 */

#ifdef HAVE_ONMD
  struct nsm_l2_oam_master *l2_oam_master;
#endif

#ifdef HAVE_CUSTOM1
#ifdef HAVE_VLAN
  /* Layer 2 related information.  */
  struct nsm_layer2_master l2;
#endif /* HAVE_VLAN */
#endif /* HAVE_CUSTOM1 */

#ifdef HAVE_VLAN
  struct nsm_vlan_access_master *vmap;
#endif /* HAVE_VLAN */

/**@brief vlan classifier type global or per-port got from hsl */
#ifdef HAVE_VLAN_CLASS
  struct hal_vlan_classifier_type *class_rule_type;
#endif /*HAVE_VLAN_CLASS*/

#ifdef HAVE_NSM_IF_PARAMS
  /* Interface parameter list.  */
  struct list *if_params;
#endif /* HAVE_NSM_IF_PARAMS */

#ifdef HAVE_LACPD
  struct bitmap *lacp_admin_key_mgr;
  struct nsm_lacp_admin_key_element *ake_list;

  /* Port Selection Criteria */
  int psc;
#endif /* HAVE_LACPD */

  struct list *phyEntityList;

  /* Disconnect time. */
  pal_time_t last_change_time;

#ifdef HAVE_QOS
  /* QoS parameter list */

  /* QoS state per VR */
  u_int8_t qos_state;

  /* MAC and IP acl list */
  struct nsm_qos_acl_master *acl;
  /* CMAP list */
  struct nsm_qos_cmap_master *cmap;

  /* PMAP list */
  struct nsm_qos_pmap_master *pmap;
#endif /* HAVE_QOS */

#ifdef HAVE_L2LERN
  struct nsm_mac_acl_master *mac_acl;
#endif /* HAVE_L2LERN */

#ifdef HAVE_L3
  struct nsm_arp_master *arp;
#endif /* HAVE_L3 */

#ifdef HAVE_IPSEC
  /* IPSEC NSM Master */
  struct nsm_ipsec_master *ipsec_master;
#endif /* HAVE_IPSEC */

#ifdef HAVE_FIREWALL
  struct nsm_firewall_master *firewall_master;
#endif /* HAVE_FIREWALL */

#ifdef HAVE_HA
  HA_CDR_REF  nsm_master_cdr_ref;

  HA_CDR_REF  nsm_master_rib_sweep_tmr_cdr_ref;
#endif /* HAVE_HA */

#ifdef HAVE_ACL
  /* list of access-list interface structure which keeps track of acl names
     configured on interfaces */
  struct list *access_group_if;
#endif /* HAVE_ACL */

  /* NSM configured and terminal debug flags.  */
  struct
  {
    struct nsm_debug_flags nmd_conf;
    struct nsm_debug_flags nmd_term;
  } nm_debug;

#ifdef HAVE_L3
#ifdef HAVE_PBR
  /* list of route map interface structure which keeps track of route maps
     configured on interfaces */
  struct list * route_map_if;
#endif /* HAVE_PBR */
```

```
#endif /* HAVE_L3 */

#ifdef HAVE_APBF
  struct list *apbf_invalid_nh;      /* IPV4 invalid NH list */
#ifdef HAVE_IPV6
  struct list *apbf_invalid_nh6;     /* IPV6 invalid NH list */
#endif /* HAVE_IPV6 */
#ifdef HAVE_APBF_NEXTHOP_PREBUILD
  struct list *nsm_apbf_nh4_tbr;     /* To Be Resolved list*/
  struct list *nsm_apbf_nh4_res;      /* Resolved list */
  struct list *nsm_apbf_nh4_nr;     /* Not Resolved list */

#ifdef HAVE_IPV6
  struct list *nsm_apbf_nh6_tbr;     /* To Be Resolved list*/
  struct list *nsm_apbf_nh6_res;      /* Resolved list */
  struct list *nsm_apbf_nh6_nr;     /* Not Resolved list */
#endif /* HAVE_IPV6 */
#endif /* HAVE_APBF_NEXTHOP_PREBUILD */
#endif /* HAVE_APBF */
};
```

pal_in4_addr

This data structure represents an IPv4 address. This data structure is defined in the `pal/dummy/pal_types.h` file.

Member	Description
`s_addr`	IPv4 address in 32-byte format

Definition

```
struct pal_in4_addr
{
  u_int32_t s_addr;
};
```

pal_in6_addr

This data structure represents an IPv6 address. This data structure is defined in the `pal/dummy/pal_types.h` file.

Type	Definition
`u6_addr8`	IPv6 address in 16-byte format
`u6_addr16`	IPv6 address in 8-byte-format
`u6_addr32`	IPv6 address in 4-byte-format

Definition

```
struct pal_in6_addr {
  union {
    u_int8_t  u6_addr8[16];
    u_int16_t u6_addr16[8];
    u_int32_t u6_addr32[4];
  } in6_u;
};
```

prefix

This data structure represents an IPv4 and IPv6 unified prefix. This data structure is defined in the `lib/prefix.h` file.

Member	Description
`family`	Prefix family
`prefixlen`	Prefix length
`prefix_style`	Prefix style
`pad1`	Padding
`prefix`	Prefix
`prefix4`	IPv4 prefix
`prefix6`	IPv6 prefix
`id`	Unspecified address family
`adv_router`	Unspecified address family
`val`	Unix domain socket address

Definition

```
struct prefix
{
  u_int8_t family;
  u_int8_t prefixlen;
  u_int8_t prefix_style;
  u_int8_t pad1;
  union
  {
    u_int8_t prefix;
    struct pal_in4_addr prefix4;
#ifdef HAVE_IPV6
    struct pal_in6_addr prefix6;
#endif /* HAVE_IPV6 */
    struct
    {
      struct pal_in4_addr id;
      struct pal_in4_addr adv_router;
    } lp;
    u_int8_t val[9];
  } u;
};
```

prefix_ipv4

This data structure represents an IPv4 address prefix. This data structure is defined in the `lib\prefix.h` file

Type	Definition
family	Address family
prefixlen	Prefix length
pad1	Padding
pad2	Padding
prefix	Address prefix

Definition

```
struct prefix_ipv4
{
  u_int8_t family;
  u_int8_t prefixlen;
  u_int8_t pad1;
  u_int8_t pad2;
  struct pal_in4_addr prefix;
}
```

prefix_ipv6

This data structure represents an IPv6 address prefix. This data structure is defined in the `lib\prefix.h` file

Type	Definition
family	Address family
prefixlen	Prefix length
pad1	Padding
pad2	Padding
prefix	Address prefix

Definition

```
struct prefix_ipv6
{
  u_int8_t family;
  u_int8_t prefixlen;
  u_int8_t pad1;
  u_int8_t pad2;
  struct pal_in6_addr prefix;
};
```

rib

This data structure represents the RIB (routing information base). This data structure is defined in the `nsm/rib/rib.h` file.

Member	Description
`next`	Next node in linked list
`prev`	Previous node in linked list
`type`	Type of this route
`sub_type`	Sub type of this route
`distance`	Distance
`flags`	Flags of this route
`metric`	Metric
`uptime`	Uptime
`ext_flags`	Extended flags of this route
`client_id`	NSM provides a four-octet client ID. To reduce memory consumption in RIB, this is defined as one octet. You can extend this member by changing its definition. The client_id is local to a system and therefore cannot be checkpointed. It is used for the graceful restart mechanism to mark the routes that are stale based on the client id. Therefore, for HA the client id will be the protocol id.
`nexthop_num`	Nexthop information
`nexthop_active_num`	Nexthop information
`nexthop`	Nexthop information
`rmm_flags`	RMM module flag
`vrf`	VRF pointer
`kernel_ms_lnode`	Kernel Msg Stagger Link-List node pointer
`nsm_rib_cdr_ref`	Checkpoint database record reference
`pid`	Process ID
`tag`	Tag
`pflags`	Inform nexthop change
`domain_conf`	OSPF Domain info

Definition

```
struct rib
```

```
{
  /* Link list. */
  struct rib *next;
  struct rib *prev;

  /* Type of this route. */
  u_char type;

  /* Sub type of this route.  */
  u_char sub_type;

  /* Distance. */
  u_char distance;

  /* Flags of this route.  */
  u_char flags;

  /* Metric */
  u_int32_t metric;

  /* Uptime. */
  pal_time_t uptime;

  /* Extended flags of this route */
  u_char ext_flags;
#define RIB_EXT_FLAG_MROUTE                 0x01
#ifdef HAVE_HA
#define RIB_EXT_FLAG_HA_RIB_CHANGED         0x02
#define RIB_EXT_FLAG_HA_RIB_DELETED         0x04
#endif /* HAVE_HA */
#define RIB_EXT_FLAG_BLACKHOLE_RECURSIVE    0x08
#define RIB_FLAG_FIB_ECMP                   0x10

  /* Client ID.  NSM protocol provide four octet client ID.  But to
     reduce memory consumption in RIB, this client_id is defined as
     one octet.  You can extend this restriction by changing this
     definition.  */
  /* XXX: Client_id is local to a system and therefore cannot be
   * checkpointed. But it is used for Graceful Restart mechanism to
   * mark the routes STALE based on client id.
   * Therefore, for HA the client id will be the protocol id. This will
   * be ensured by assigning the client_id as the protocol_id at time
   * of NSM client connect (in nsm_server_recv_service() ).
   */
  u_char client_id;

  /* Nexthop information. */
  u_char nexthop_num;
  u_char nexthop_active_num;
  struct nexthop *nexthop;
#ifdef HAVE_RMM
  /* RMM module flag.  */
  u_char rmm_flags;
#endif /* HAVE_RMM */

  /* VRF pointer.  */
  struct nsm_vrf *vrf;

#ifdef HAVE_STAGGER_KERNEL_MSGS
  /* Kernel Msg Stagger Link-List node pointer. */
  struct listnode *kernel_ms_lnode;
#endif /* HAVE_STAGGER_KERNEL_MSGS */

#ifdef HAVE_HA
  HA_CDR_REF nsm_rib_cdr_ref;
#endif /* HAVE_HA */

 /*Process ID */
 u_int32_t pid;

 /* Tag */
 u_int32_t tag;

 /* inform nexthop change */
 u_int32_t pflags;
#define NSM_ROUTE_CHG_INFORM_BGP (1 << 0)
#ifdef HAVE_BFD
#define NSM_ROUTE_CHG_BFD (1 << 1)
#define NSM_BFD_CONFIG_CHG (1 << 2)
#endif /* HAVE_BFD */
#ifdef HAVE_MPLS
#define NSM_ROUTE_HAVE_IGP_SHORTCUT (1 << 3)
#endif/* HAVE_MPLS */
#ifdef HAVE_VRF
 /*OSPF Domain info */
 struct nsm_ospf_domain_conf *domain_conf;
#endif /*HAVE_VRF*/
};
```

stream

This data structure represents a fixed-length buffer for network output/input. This data structure is defined in the `lib/stream.h` file.

Member	Description
`next`	Next node in linked list
`data`	Previous node in linked list
`putp`	Put pointer
`getp`	Get pointer
`endp`	End of pointer
`size`	Data size

Definition

```
struct stream
{
  struct stream *next;
  u_char *data;
  /* Put pointer. */
  u_int32_t putp;
  /* Get pointer. */
  u_int32_t getp;
  /* End of pointer. */
  u_int32_t endp;
  /* Data size. */
  u_int32_t size;
};
```

thread

This data structure represents a thread. This data structure is defined in the `lib/thread.h` file.

Member	Description
next	Linked list
prev	Linked list
master	Pointer to the struct thread_master
zg	Pointer to the struct lib_globals
func	Event function
arg	Event argument
type	Thread type
priority	Priority
index	Thread timer index
u	Arguments
stime	HA timer checkpoint: Timer absolute starting time
period	HA timer checkpoint: Timer period

Definition

```
struct thread
{
  /* Linked list.  */
  struct thread *next;
  struct thread *prev;

  /* Pointer to the struct thread_master.  */
  struct thread_master *master;

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

 /* Event function.  */
  int (*func) (struct thread *);

  /* Event argument.  */
  void *arg;

  /* Thread type.  */
  char type;
  /* Priority.  */
  char priority;
#define THREAD_PRIORITY_HIGH         0
#define THREAD_PRIORITY_MIDDLE       1
#define THREAD_PRIORITY_LOW          2

  /* Thread timer index.  */
  char index;

  /* Arguments.  */
  union
  {
    /* Second argument of the event.  */
    int val;

    /* File descriptor in case of read/write.  */
    int fd;
  /* Rest of time sands value.  */
    struct pal_timeval sands;
  } u;

#ifdef HAVE_HA
  /* Additions for the HA dependent processing. */
  /* This values will go with the timer checkpoint. */
  struct pal_timeval stime;  /* Timer's absolute starting time. */
  struct pal_timeval period; /* Timer's period. */
#endif /* HAVE_HA */
};
```

variable

This data structure represents an SNMP variable. This data structure is defined in the `lib/snmp.h` file.

Member	Description
magic	Index of the MIB
type	Type of variable
acl	Access control list
findVar	Callback function
namelen	Suffix of the MIB
lg	Library globals

Definition

```
struct variable
{
  /* Index of the MIB.*/
  u_int8_t magic;

  /* Type of variable. */
  char type;

  /* Access control list. */
  u_int8_t acl;

  /* Callback function. */
  FindVarMethod *findVar;

  /* Suffix of the MIB. */
  u_int8_t namelen;
  oid name[MAX_OID_LEN];

  /* Lib globals */
  struct lib_globals *lg;
};
```