ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Simple Management Interface Client 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 shows how to write Simple Management Interface (SMI) client applications.

Audience

This guide is intended for developers who write SMI client applications that configure and manage 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 document contains these chapters:

Related Documents

Use this guide with the corresponding SMI API reference for the protocols/features for which you are writing client applications. Each SMI reference manual has a file name with this format:

- ZebOS-XP-SMI-Reference-*xxx*.pdf

where *xxx* is the name of the protocol or feature. For example, the SMI reference manual for BGP is:

- ZebOS-XP-SMI-Reference-BGP.pdf

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 About the Simple Management Interface

This chapter introduces the Simple Management Interface (SMI). SMI is a C language API (Application Programming Interface) that you use to write applications that configure and manage the ZebOS-XP routing and switching protocols.

Every ZebOS-XP protocol module provides a C language API for configuring properties of the protocol. For example, the `ospf_api.h` include file defines an API for configuring the OSPF protocol. However, you must call the functions in a ZebOS-XP C language API in the same process space as ZebOS-XP (that is, from a ZebOS-XP daemon). SMI provides a mechanism to call the equivalent functions from a *different* process space.

SMI provides get and set operations that duplicate operations you can perform in `imish`. In this respect, SMI operations are similar to SNMP get and set operations. SMI also supports asynchronous event notification that is similar to the SNMP trap mechanism.

SMI uses a client-server architecture. The SMI client is a dynamically linked object library. An SMI server runs as part of each ZebOS-XP protocol module daemon. An SMI server starts automatically whenever its daemon starts. An SMI client uses socket-based inter-process communication (IPC) to exchange messages with the SMI server.

Your application makes calls to functions in the SMI client that internally creates a message and sends it to the SMI server. The SMI server calls functions in the ZebOS-XP API and returns the result to the client. Figure 1-1 shows the high-level architecture of the SMI framework.

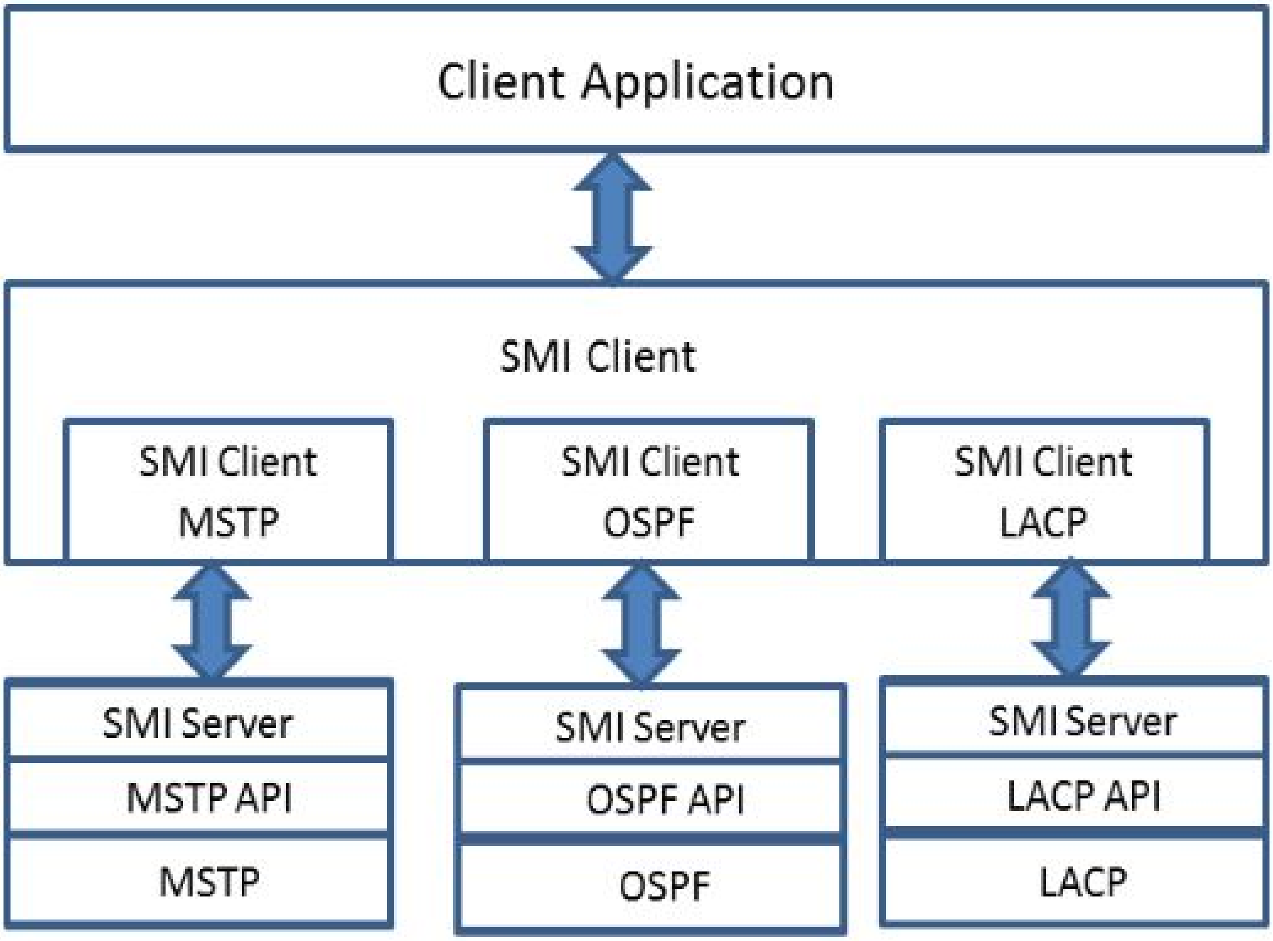

Figure 1-1: SMI architecture

CHAPTER 2 Writing an SMI Client

This chapter shows how to create, start, stop and delete an SMI client.

Process Flow

Figure 2-1 shows the steps that you follow in an SMI client application.

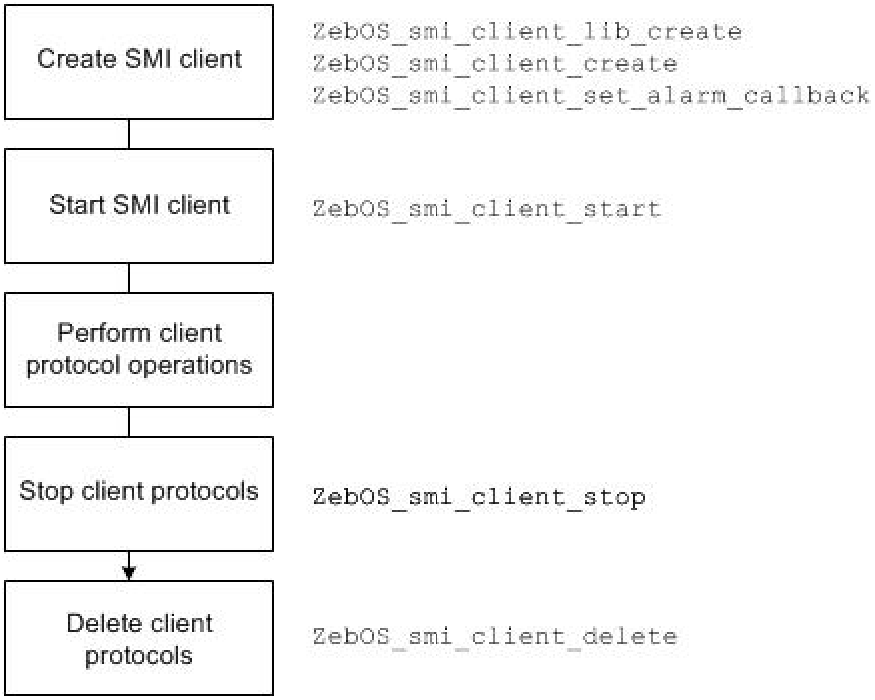

Figure 2-1: SMI client application process flow

Creating an SMI Client

To create an SMI client in your application:

1. Call ZebOS_smi_client_lib_create to initialize the global SMI client structure
2. Call ZebOS_smi_client_create to create the client protocol modules that your application manages
3. Optionally, call ZebOS_smi_client_set_alarm_callback to register a callback function that is invoked when the SMI server sends alarms; for more, Chapter 4, *Handling Alarms*

For example:

```
struct smiclient_globals *azg;
char lsr_name [SMI_LSR_NAMSIZ];
lsr_name [0] = '\0';
azg = ZebOS_smi_client_lib_create(PAL_TRUE, lsr_name);
```

```
ZebOS_smi_client_create (azg, SMI_AC_ALL, SMI_AC_DEBUG);
```

Starting a Client

After creating an API client, your application must start client protocol modules by calling ZebOS_smi_client_start.

For example:

```
unsigned int errflag = 0;
char *ipaddr = NULL;    // From command line
char ipaddr_str[16];
pal_snprintf (ipaddr_str, sizeof (ipaddr_str), argv[1]);
ipaddr = ipaddr_str;
errflag = ZebOS_smi_client_start(azg, ipaddr);
```

Performing Operations

After you have started a client protocol, you can call its functions. Each function requires the global SMI client structure as a parameter. The protocol client functions are described in their respective reference manuals.

For example, the code below enables OSPF routing on an interface. This code performs the same operation as the `ospf6_if_ipv6_router_set` function in the ZebOS-XP C API or the `ipv6 router ospf area` command.

```
int ret= -1;
u_int32_t vr_id = 0;
char  name[255];
sprintf(name, "%s", "eth11");
struct pal_in4_addr area_id;
char ip_area_id[255];
sprintf(ip_area_id, "%s", "10.12.7.5");
buildip(ip_area_id, &area_id);
int format = 1;
char tag[255];
sprintf(tag, "%s", "WORD");
int instance_id = 10;
ret = smi_ospf6_if_ipv6_router_set(azg, vr_id, name, area_id, format, tag, 
instance_id);
```

Stopping a Client

You can stop a client protocol with the ZebOS_smi_client_stop function. Stopping a client protocol temporarily stops the delivery of incoming messages. However, stopping a client does not stop the SMI server from sending messages.

For example, to stop the NSM client protocol:

```
ZebOS_smi_client_stop(azg, SMI_AC_NSM_MODULE);
```

You can restart a client protocol later using ZebOS_smi_client_start.

Deleting a Client

After stopping a client, you can delete it with the `ZebOS_smi_client_delete` function. Deleting a client essentially severs the connection between the SMI server and the SMI client. You must stop a client with ZebOS_smi_client_stop before you delete the client.

For example, to delete the NSM client protocol:

```
ZebOS_smi_client_delete(&azg, SMI_AC_NSM_MODULE);
```

Linking to the SMI Shared Library

When you build ZebOS-XP, the SMI client is created as a dynamically linked object library in `platform/linux/bin/smi.so`. You must link your client application to the `smi.so` library.

In a production environment, you need to set up `smi.so` using standard shared object management techniques.

CHAPTER 3 Client API Reference

The chapter describes the functions that you call in a SMI client application.

The functions in this chapter are used to create the client environment and establish a connection between client and server.

Data Structures and Enumerations

This section describes the data structures and enumerations used by the SMI client functions.

smiclient_globals

This structure represents the SMI client global data structure and is defined in `lib/smi/client/smi_client.h`.

Type	Definition
`cindex`	A bit mask used to identify the attributes that are filled in this structure
`smi_zg`	Global data structure to maintain data per protocol demon (used for internal operations)
`ac`	SMI client data structure (used for internal operations)
`client_type`	SMI client type (used for internal operations)
`remote_addr`	IP address of the SMI server
`smi_err_str`	Error string sent from the SMI server to SMI client
`lsr_name`	Logical switch router name
`debug`	Flag indicating if debugging is enabled or disabled

Definition

```
struct smiclient_globals {
  u_int32_t cindex;
  struct lib_globals *smi_zg;
  struct smi_client *ac[SMI_AC_MAX];
  smi_client_type client_type;
  char   *remote_addr; /* For remote connections */
  char smi_err_str [SMI_ERRMSG_BUF_SIZE+1];
  /*! LSR name */
  char lsr_name[SMI_LSR_NAMSIZ + 1];
  int debug;
};
```

smi_api_module

This enumeration in `lib/smi/client/smi_message.h` specifies protocol client identifiers.

Name	Protocol/component
SMI_AC_8021X_MODULE	802.1x
SMI_AC_ALL	All protocol clients
SMI_AC_BGP_MODULE	BGP
SMI_AC_DVMRP_MODULE	DVMPR
SMI_AC_HOSTP_MODULE	Host protocol
SMI_AC_ISIS_MODULE	ISIS
SMI_AC_LACP_MODULE	LACP
SMI_AC_L2MRIB_MODULE	Layer 2 Multicast RIB
SMI_AC_LSM_MODULE	LSM
SMI_AC_MSTP_MODULE	MSTP
SMI_AC_MRIB_MODULE	Multicast RIB
SMI_AC_NDD_MODULE	NDD
SMI_AC_NSM_MODULE	NSM
SMI_AC_OAM_MODULE	OAM
SMI_AC_ONM_MODULE	ONM
SMI_AC_OSPF_MODULE	OSPF
SMI_AC_OSPF6_MODULE	OSPF6
SMI_AC_PIM_MODULE	PIM
SMI_AC_PTP_MODULE	PTP
SMI_AC_RIB_MODULE	RIB
SMI_AC_RIP_MODULE	RIP
SMI_AC_RIPNG_MODULE	RIPng
SMI_AC_RMON_MODULE	RMON
SMI_AC_VPORTMGR_MODULE	VPORT
SMI_AC_VRRP_MODULE	VRRP

Definition

```
typedef enum _smi_api_module {
  SMI_AC_NSM_MODULE,
  SMI_AC_LACP_MODULE,
  SMI_AC_MSTP_MODULE,
  SMI_AC_RMON_MODULE,
  SMI_AC_ONM_MODULE,
  SMI_AC_VPORTMGR_MODULE,
  SMI_AC_OSPF_MODULE,
#ifdef HAVE_HOSTPD
  SMI_AC_HOSTP_MODULE,
#endif /* HAVE_HOSTPD */
  SMI_AC_OSPF6_MODULE,
  SMI_AC_8021X_MODULE,
  SMI_AC_OAM_MODULE,
  SMI_AC_ISIS_MODULE,
  SMI_AC_MRIB_MODULE,
  SMI_AC_PTP_MODULE,
  SMI_AC_DVMRP_MODULE,
  SMI_AC_RIP_MODULE,
  SMI_AC_RIPNG_MODULE,
  SMI_AC_PIM_MODULE,
  SMI_AC_BGP_MODULE,
  SMI_AC_VRRP_MODULE,
  SMI_AC_RIB_MODULE,
  SMI_AC_LSM_MODULE,
  SMI_AC_L2MRIB_MODULE,
#ifdef HAVE_NDD
  SMI_AC_NDD_MODULE,
#endif /* HAVE_NDD */
  SMI_AC_MAX,
  SMI_AC_API_CLIENT,
} smi_api_module;
```

Include File

To call the functions in this chapter, you must include `lib\smi\client\smi_client.h`.

API Reference

This section describes each SMI client function.

The following table list the functions for SMI client applications. To navigate to the topic for a function, click its hyperlink.

API Function	Description
ZebOS_smi_client_lib_create	Allocates and initializes memory for the SMI client
ZebOS_smi_client_create	Creates a protocol client
ZebOS_smi_client_start	Starts protocol clients

API Function	Description
ZebOS_smi_client_stop	Closes the client connection
ZebOS_smi_client_delete	Cancels pending read messages and releases memory allocated for the client
ZebOS_smi_client_set_alarm_callback	Registers an alarm callback function

ZebOS_smi_client_lib_create

This function allocates and initializes memory for an SMI client application and returns a pointer to the global SMI client structure that must be passed as a parameter in all later function calls. This is the first function that you must call in an SMI client application.

Syntax

```
struct smiclient_globals *
ZebOS_smi_client_lib_create (bool_t set_mutex, char *lsr_name)
```

Input Parameters

set_mutex	Whether the client application is single threaded or multi-threaded:
PAL_TRUE	Client application is multi-threaded and SMI must sequence all requests
PAL_FALSE	Client application is single threaded
lsr_name	Logical switch router name

Return Values

Pointer to smiclient_globals when the function succeeds

NULL when the function falis

ZebOS_smi_client_create

This function creates a protocol client.

Syntax

```
int
ZebOS_smi_client_create (struct smiclient_globals * azg,
                         smi_api_module module, int debug
```

Input parameters

azg	Pointer to smiclient_globals structure
module	Protocol client to create as shown in smi_api_module
debug	Whether debugging is enabled:
PAL_FALSE	Disable debugging
PAL_TRUE	Enable debugging

Output parameters

None

Return values

`SMI_SUCCESS` when the function succeeds

`SMI_INVALID_VAL` when `module` is not a valid value

ZebOS_smi_client_start

This function starts all protocol clients that have been created and initiates a socket connection with the SMI server for each protocol module.

Note: You must call ZebOS_smi_client_create before calling the this function.

Syntax

```
int
ZebOS_smi_client_start(struct smiclient_globals *azg, char *ipaddr)
```

Input parameters

`azg`	Pointer to the smiclient_globals structure
`ipaddr`	IP address of the SMI server that is running a protocol module: • Dotted decimal notation (for example, A.B.C.D) if the SMI server and client are running remotely (on separate systems) • `NULL` if the SMI client and server are running on the same machine

Output parameters

None

Return values

0 when the function succeeds

`SMI_AC_8021X_INITERR` when the 802.1x SMI client start fails

`SMI_AC_BGP_INITERR` when the BGP SMI client start fails

`SMI_AC_HOSTP_INITERR` when the host protocol SMI client start fails

`SMI_AC_ISIS_INITERR` when the ISIS SMI client start fails

`SMI_AC_L2MRIB_INITERR` when the Layer 2 Multicast RIB SMI client start fails

`SMI_AC_LACP_INITER` when the LACP SMI client start fails

`SMI_AC_LSM_INITERR` when the LSM SMI client start fails

`SMI_AC_MRIB_INITERR` when the Multicast RIB SMI client start fails

`SMI_AC_MSTP_INITERR` when the MSTP SMI client start fails

`SMI_AC_NDD_INITERR` when the NDD SMI client start fails

`SMI_AC_NSM_INITERR` when the NSM SMI client start fails

`SMI_AC_OAM_INITERR` when the OAM SMI client start fails

`SMI_AC_ONM_INITERR` when the ONM SMI client start fails

`SMI_AC_OSPF_INITERR` when the OSPF SMI client start fails

`SMI_AC_OSPF6_INITERR` when the OSPF6 SMI client start fails

`SMI_AC_PIM_INITERR` when the PIM SMI client start fails

`SMI_AC_PTP_INITERR` when the PTP SMI client start fails

`SMI_AC_RIB_INITERR` when the unicast RIB SMI client start fails

`SMI_AC_RIP_INITERR` when the RIP SMI client start fails

`SMI_AC_RIPNG_INITERR` when the RIPng SMI client start fails

`SMI_AC_RMON_INITERR` when the RMON SMI client start fails

`SMI_AC_VPORTMGR_INITERR` when the VPORT SMI client start fails

`SMI_AC_VRRP_INITERR` when the VRRP SMI client start fails

ZebOS_smi_client_stop

This function stops a client protocol and closes the ZebOS-XP client socket.

Stopping a client protocol temporarily stops the delivery of incoming messages. However, stopping a client does not stop the SMI server from sending messages.

Syntax

```
int
ZebOS_smi_client_stop (struct smiclient_globals *azg, int module)
```

Input parameters

`azg`	Pointer to the smiclient_globals structure
`module`	Protocol client to stop as shown in smi_api_module. You must stop each protocol client separately; you cannot specify `SMI_AC_ALL`.

Output parameters

None

Return values

`SMI_SUCCESS` when the function succeeds

`SMI_INVALID_VAL` when `module` is not a valid value

ZebOS_smi_client_delete

This function deletes a client, cancelling all pending read messages, and releasing memory allocated for the client.

Note: You must call ZebOS_smi_client_stop before calling the this function.

Syntax

```
int
ZebOS_smi_client_delete (struct smiclient_globals **azg, int module)
```

Input parameters

`azg`	Double pointer to the smiclient_globals structure

`module`	Protocol client to stop as shown in smi_api_module. You must delete each protocol client separately; you cannot specify `SMI_AC_ALL`.

Output parameters

None

Return values

`SMI_SUCCESS` when the function succeeds

`SMI_INVALID_VAL` when `module` is not a valid value

ZebOS_smi_client_set_alarm_callback

This function registers a callback function that is called when alarms are generated from the SMI server.

For more about SMI alarms, see Chapter 4, *Handling Alarms*.

Note: You must call ZebOS_smi_client_create before calling this function.

Syntax

```
void
ZebOS_smi_client_set_alarm_callback (smi_alarm_callback_t callback)
```

Input parameters

`callback`	Callback function that is invoked at the SMI client side when an alarm is received from the SMI server; see smi_alarm_callback_t

Output parameters

None

Return values

None

CHAPTER 4 Handling Alarms

This section explains the code elements you use to handle alarms in an SMI client application.

About SMI Alarms

To handle notification about alarms from the SMI server, you must:

- Define a callback function in the SMI client application whose signature match the typedef smi_alarm_callback_t.
- Register your callback function with the ZebOS_smi_client_set_alarm_callback function

In the implementation of the callback, you can:

- Determine the alarm identifier as shown in smi_alarm
- Determine the protocol client as shown in smi_api_module
- Get details about the alarm from the smi_msg_alarm structure

smi_alarm_callback_t

This typedef defines a signature for a callback function that you write to process an alarm received from the SMI server.

Note: You must register this function with the ZebOS_smi_client_set_alarm_callback.

Syntax

```
typedef void
(* smi_alarm_callback_t) (smi_alarm alarm, smi_api_module module, void *data)
```

Input parameters

`alarm`	Alarm identifier as shown in smi_alarm
`module`	Protocol client as shown in smi_api_module
`data`	Alarm data as shown in smi_msg_alarm; depending on the type of alarm, the SMI server fills the data in the corresponding member variables of this structure

Output parameters

None

Return values

None

Data Structures and Enumerations

The objects in this section are defined in the `smi/client/smi_message.h` file.

smi_msg_alarm

This structure provides data about an alarm that is passed to the alarm handler function you assign with ZebOS_smi_client_set_alarm_callback. Depending on the type of alarm generated, the SMI server fills the data in the corresponding member variables of this structure.

Type	Definition
cindex	A bit mask used to identify the attributes that are filled in this structure
smi_module	One of the constants from the smi_api_module enumeration
alarm_type	One of the constants from the smi_alarm enumeration
nsm_client	Data for SMI_ALARM_NSM_CLIENT_SOCKET_DISCONNECT, SMI_ALARM_SMI_SERVER_CONNECT, and SMI_ALARM_SMI_SERVER_DISCONNECT
description	Data for SMI_ALARM_TRANSPORT_FAILURE
cfm_alarm_info	Data for SMI_ALARM_CFM
efm_alarm_info	Data for SMI_ALARM_EFM
stp_alarm_info	Data for SMI_ALARM_STP
rmon_alarm_info	Data for SMI_ALARM_RMON
loc_alarm_info	Data for SMI_ALARM_LOC
vlan_alarm_info	Data for SMI_ALARM_NSM_VLAN_ADD_TO_PORT, SMI_ALARM_NSM_VLAN_DEL_FROM_PORT, and SMI_ALARM_NSM_VLAN_PORT_BULK_UPDATE
vlan_port_mode_alarm_info	Data for SMI_ALARM_NSM_VLAN_PORT_MODE
bridge_proto_change_alarm_info	Data for SMI_ALARM_NSM_BRIDGE_PROTO_CHANGE

Definition

```
struct smi_msg_alarm
{
  smi_cindex_t cindex;
#define SMI_ALARM_CTYPE_MODULE_NAME                      0
#define SMI_ALARM_CTYPE_ALARM_TYPE                       1
#define SMI_ALARM_CTYPE_DATA_NSM_CLIENT                  2
#define SMI_ALARM_CTYPE_DATA_TRANSPORT_DESC              3
#define SMI_ALARM_CTYPE_DATA_CFM_ALARM                   4
#define SMI_ALARM_CTYPE_DATA_EFM_ALARM                   5
#define SMI_ALARM_CTYPE_DATA_STP_ALARM                   6
#define SMI_ALARM_CTYPE_DATA_RMON_ALARM                  7
#define SMI_ALARM_CTYPE_LOC_ALARM                        8
#define SMI_ALARM_CTYPE_VLAN_ALARM                       9
#define SMI_ALARM_CTYPE_VLAN_PORT_MODE_ALARM             10
#define SMI_ALARM_CTYPE_BRIDGE_PROTOCOL_CHANGE_ALARM     11
  smi_api_module smi_module;
  smi_alarm alarm_type;
  /* data for SMI_ALARM_NSM_CLIENT_SOCKET_DISCONNECT,
   * SMI_ALARM_SMI_SERVER_CONNECT
   * and SMI_ALARM_SMI_SERVER_DISCONNECT
   */
  smi_nsm_client nsm_client;

  /* data for SMI_ALARM_ SMI_ALARM_TRANSPORT_FAILURE */
#define SMI_TRANSPORT_DESC_MAX 512
  u_char description [SMI_TRANSPORT_DESC_MAX];

  /* data for SMI_ALARM_CFM */
  struct smi_cfm_alarm_info cfm_alarm_info;

  /* data for SMI_ALRM_EFM */
  struct smi_efm_alarm_info efm_alarm_info;

  /* data for SMI_ALARM_STP */
  struct smi_stp_alarm_info stp_alarm_info;

  /* data for SMI_ALARM_RMON */
  struct smi_rmon_alarm_info rmon_alarm_info;

  /* data for SMI_ALARM_LOC */
  struct smi_loc_alarm_info loc_alarm_info;

  /* data for */
  struct smi_vlan_port_alarm vlan_alarm_info;

  struct smi_vlan_port_mode_alarm vlan_port_mode_alarm_info;

  struct smi_bridge_protocol_change_alarm bridge_proto_change_alarm_info;
};
```

smi_alarm

This enumeration defines alarm identifiers generated by the SMI server.

Type	Definition
SMI_ALARM_MEMORY_FAILURE	Memory allocation failed
SMI_ALARM_HARDWARE_FAILURE	Not used
SMI_ALARM_NSM_SERVER_SOCKET_DISCONNECT	Socket connection between the protocol module and NSM is disconnected

Type	Definition
SMI_ALARM_NSM_CLIENT_SOCKET_DISCONNECT	Socket connection disconnected between NSM SMI client and server
SMI_ALARM_TRANSPORT_FAILURE	Socket connection with HSL disconnected
SMI_ALARM_CFM	Alarm generated by CFM module
SMI_ALARM_EFM	Alarm generated by EFM module
SMI_ALARM_STP	Alarm generated by STP module
SMI_ALARM_RMON	Alarm generated by RMON module
SMI_ALARM_LOC	NSM generated the interface down update event
SMI_ALARM_SMI_SERVER_CONNECT	Successful socket connection between the API client and server
SMI_ALARM_SMI_SERVER_DISCONNECT	Socket connection disconnected between the client and server
SMI_ALARM_NSM_VLAN_ADD_TO_PORT	Alarm generated by the NSM module when a VLAN is added
SMI_ALARM_NSM_VLAN_DEL_FROM_PORT	Alarm generated by the NSM module when a VLAN is deleted
SMI_ALARM_NSM_VLAN_PORT_MODE	Alarm indicates that the port mode is set for the aggregated ports of type smi_vlan_port_mode
SMI_ALARM_NSM_BRIDGE_PROTO_CHANGE	Alarm is raised when the bridge type is changed, such as from STP to RSTP
SMI_ALARM_NSM_VLAN_PORT_BULK_UPDATE	Alarm is raised when bulk update fails when number of ports are added to VLAN

Definition

```
typedef enum _smi_alarm {
  SMI_ALARM_MEMORY_FAILURE,
  SMI_ALARM_HARDWARE_FAILURE,
  SMI_ALARM_NSM_SERVER_SOCKET_DISCONNECT,
  SMI_ALARM_NSM_CLIENT_SOCKET_DISCONNECT,
  SMI_ALARM_TRANSPORT_FAILURE,
  SMI_ALARM_CFM,
  SMI_ALARM_EFM,
  SMI_ALARM_STP,
  SMI_ALARM_RMON,
  SMI_ALARM_LOC,
  SMI_ALARM_SMI_SERVER_CONNECT,
  SMI_ALARM_SMI_SERVER_DISCONNECT,
  SMI_ALARM_NSM_VLAN_ADD_TO_PORT,
  SMI_ALARM_NSM_VLAN_DEL_FROM_PORT,
  SMI_ALARM_NSM_VLAN_PORT_MODE,
  SMI_ALARM_NSM_BRIDGE_PROTO_CHANGE,
  SMI_ALARM_NSM_VLAN_PORT_BULK_UPDATE,
  SMI_ALARM_SMI_MAX
} smi_alarm;
```

smi_nsm_client

This enum defines the protocol module identifiers that interact with NSM.

Type	Definition
SMI_NSM_CLIENT_LACP	IPI_PROTO_LACP
SMI_NSM_CLIENT_MSTP	IPI_PROTO_MSTP
SMI_NSM_CLIENT_IMI	IPI_PROTO_IMI
SMI_NSM_CLIENT_RMON	IPI_PROTO_RMON
SMI_NSM_CLIENT_ONM	IPI_PROTO_ONM
SMI_NSM_CLIENT_VPORTMGR	IPI_PROTO_VPORTMGR
SMI_NSM_CLIENT_MAX	Maximum protocol value

Definition

```
typedef enum _smi_nsm_client {
  SMI_NSM_CLIENT_LACP = 15,    /* IPI_PROTO_LACP */
  SMI_NSM_CLIENT_MSTP = 18,    /* IPI_PROTO_MSTP */
  SMI_NSM_CLIENT_IMI  = 19,    /* IPI_PROTO_IMI  */
  SMI_NSM_CLIENT_RMON = 24,    /* IPI_PROTO_RMON */
  SMI_NSM_CLIENT_ONM  = 25,    /* IPI_PROTO_ONM  */
  SMI_NSM_CLIENT_VPORTMGR  = 38,    /* IPI_PROTO_VPORTMGR  */
  SMI_NSM_CLIENT_MAX  = 39     /* IPI_PROTO_MAX  */
} smi_nsm_client;
```

smi_cfm_alarm_info

This structure defines a CFM alarm.

Type	Definition
md_name	MD name
level	MD level
ma_name	MA Name
vid	VLAN on which the fault was detected
mep_id	MEP identifier
mep_dir	MEP direction

Type	Definition
ifname	Interface name
mac_add	MAC address
flags	Flags

Definition

```
struct smi_cfm_alarm_info
{
  u_char md_name[SMI_MD_NAME_LENGTH];

  /* MD Level */
  u_int32_t level;

  /* MA Name */
  u_char ma_name[SMI_MA_NAME_LENGTH];

  /* VLAN on which the the fault was detected */
  u_int16_t vid;

  /* MEP ID */
  u_int32_t mep_id;

  /* MEP Direction */
  enum smi_cfm_mep_dir mep_dir;

  /* Interface Name*/
  char ifname [SMI_INTERFACE_NAMSIZ + 1];

  /* MAC address */
  char mac_add [SMI_ETHER_ADDR_LEN];

  /*
   * SMI_MEP_FAULT    1 << 1 : MEP Has detected a fault
   * SMI_MA_RDI       1 << 2 : RDI has been detected in MA
   * SMI_MAC_TLV_ERR  1 << 3 : Some Remote MEP notified MAC status error
   * SMI_CCM_ERR      1 << 4 : Atleast one remote MEP is not transmitting CCM
   * SMI_XCON_ERR     1 << 5 : Received a CCM from MEP in different MA
   */
  u_int8_t flags;
}
```

smi_efm_alarm_info

This structure defines a EFM alarm.

Type	Definition
`ifname`	Interface name
`flags`	Flags

Definition

```
struct smi_efm_alarm_info
{
  /* Interface Name*/
  char ifname [SMI_INTERFACE_NAMSIZ + 1];

#define SMI_EFM_REM_DYING_GASP 1    <<  1 /* Remote OAM client
                                          * detected dying gasp */

#define SMI_EFM_LOC_DYING_GASP 1    <<  2 /* Local OAM client
                                          * detected dying gasp */

#define SMI_EFM_REM_CRIT_EVENT 1    <<  3 /* Remote OAM client
                                          * detected
                                          * critical event */

#define SMI_EFM_LOC_CRIT_EVENT 1    <<  4 /* Local OAM client detected
                                          * critical event */

#define SMI_EFM_REM_LINK_FAULT 1    <<  5 /* Remote OAM client
                                          * detected
                                          * Link Fault */

#define SMI_EFM_LOC_LINK_FAULT 1    <<  6 /* Local OAM client detected
                                          * link fault*/

#define SMI_EFM_LINK_LOST      1    <<  7 /* Local OAM client detected
                                          * that remote OAM client
                                          * is no longer sending
                                          * OAM PDUs
                                          */
#define SMI_EFM_LOOPBACK_ON    1    <<  8 /* Remote loopback on */
#define SMI_EFM_LOOPBACK_OFF   1    <<  9 /* Remote loopback off */
  u_int32_t flags;

};
```

smi_stp_alarm_info

This structure defines a STP alarm.

Type	Definition
`ifname`	Interface name
`flags`	Flags

Definition

```
struct smi_stp_alarm_info
{
  /* Interface Name*/
  char ifname [SMI_INTERFACE_NAMSIZ + 1];

  /* STP Detected a BPDU Guard Violation */
#define SMI_STP_BPDU_GUARD_VIOLATE_SET          1  <<  0

  /* STP Detected a Root Guard Violation */
#define SMI_STP_ROOT_GUARD_VIOLATE_SET          1  <<  1

  /* STP Detected a BPDU filter Violation */
#define SMI_STP_BPDU_FILTER_VIOLATE_SET         1  <<  2

  /* STP Resets a BPDU Guard Violation Alarm*/
  #define SMI_STP_BPDU_GUARD_VIOLATE_UNSET      1  <<  3

  /* STP Resets a Root Guard Violation Alarm*/
#define SMI_STP_ROOT_GUARD_VIOLATE_UNSET        1  <<  4

  /* STP Resets a BPDU Filter Violation Alarm*/
#define SMI_STP_BPDU_FILTER_VIOLATE_UNSET       1  <<  5

  u_int8_t flags;
};
```

smi_rmon_alarm_info

This structure provides data about an RMON alarm.

Type	Definition
`ifname`	Interface name
`etherStatObjName`	The rmonEtherStatsGroup object that generated the alarm (such as etherStatsOversizePkts)

Type	Definition
alarmSampleType	Method of sampling the selected variable and calculating the value to compare against the thresholds: • If SMI_ALARM_ABS, the value of the selected variable is compared directly with the thresholds at the end of the sampling interval. • If SMI_ALARM_DELTA, the value of the selected variable at the last sample is subtracted from the current value, and the difference compared with the thresholds.
alarm_type	Indicates whether a rising threshold (SMI_ALARM_RISING_THRESHOLD) alarm or falling threshold (SMI_ALARM_RISING_THRESHOLD) alarm
threshold	Value of the raising threshold
current_counter_value	Current value of the counter triggering the alarm

Definition

```
struct smi_rmon_alarm_info
{
  char ifname [INTERFACE_NAMSIZ + 1];
  char etherStatObjName [SMI_RMON_ALARM_VAR_WORD_LENGTH + 1];
#define SMI_ALARM_DELTA 0
#define SMI_ALARM_ABS   1
  u_int8_t alarmSampleType;
#define SMI_ALARM_RISING_THRESHOLD  1
#define SMI_ALARM_FALLING_THRESHOLD 0
  u_int8_t alarm_type;
  ut_int64_t thresHold;
  ut_int64_t current_counter_value;
}
```

smi_loc_alarm_info

This structure provided the data about an interface down alarm.

Type	Definition
ifname	Interface name

Definition

```
struct smi_loc_alarm_info
{
  char ifname [INTERFACE_NAMSIZ + 1];
}
```

smi_vlan_port_alarm

This structure provides information about an alarm when a VLAN is added.

Type	Definition
ifname	Interface name
vlan_bmp	VLAN IDs to add
egr_bmp	Egress VLAN IDs to add
bulk_alarm	Bulk alarm

Definition

```
struct smi_vlan_port_alarm
{
  char ifname [INTERFACE_NAMSIZ + 1];
  struct smi_vlan_bmp vlan_bmp;
  struct smi_vlan_bmp egr_bmp;
  struct smi_vlan_port_list_bulk_alarm bulk_alarm;
};
```

smi_vlan_port_mode_alarm

This structure provides information about an alarm whenever a VLAN added.

Type	Definition
ifname	Interface name or port where the port mode change is reported
mode	VLAN port mode of type smi_vlan_port_mode
sub_mode	VLAN port sub mode of type smi_vlan_port_mode

Definition

```
struct smi_vlan_port_mode_alarm
{
  char ifname [INTERFACE_NAMSIZ + 1];
  enum smi_vlan_port_mode mode;
  enum smi_vlan_port_mode sub_mode;
};
```

smi_bridge_protocol_change_alarm

This structure provides information about a bridge type change, such as from STP to RSTP.

Type	Definition
brname	Bridge name
type	Bridge type
topo_type	Topology type

Definition

```
struct smi_bridge_protocol_change_alarm
{
  char brname [SMI_BRIDGE_NAMSIZ + 1];
  enum smi_bridge_type type;
  enum smi_bridge_topo_type topo_type;
};
```

Index