ZebOS-XP®
Network Platform

Version 1.4
Extended Performance

Ethernet Local Management Interface
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 Ethernet Local Management Interface (ELMI).

Audience

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

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, *Ethernet Local Management Interface*
- Chapter 2, *System Architecture*
- Chapter 3, *System Messaging*
- Chapter 4, *Data Structures*
- Chapter 5, *ELMI Command API*
- Chapter 6, *ELMI Module API*
- Appendix A, *System Messaging Details*
- Appendix B, *System Services*

Related Documents

The following guides are related to this document:

- *Ethernet Local Management Interface Command Reference*

- *Ethernet Local Management Interface Configuration Guide*
- *Installation Guide*
- *Network Services Module Developer Guide*
- *Network Services Module Command Reference*
- *Architecture Guide*

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

Support

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

Comments

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

CHAPTER 1 Ethernet Local Management Interface

Ethernet Local Management Interface (ELMI) is a portable software solution designed for the Metro Ethernet Forum 16 (MEF16) specification.

Overview

ELMI is an Ethernet OAM (Operation, Administration, and Management) protocol used for communications between two User Network Interfaces (that is, UNI-C and UNI-N). ELMI provides both UNI and EVC (Ethernet Virtual Connection) status information to customer edge (CE) devices. This information enables automatic configuration of CE operation based on the Metro Ethernet Network configuration.

ELMI includes the following procedures according to the MEF16 specifications:

- Notifies a CE device when an Ethernet Virtual Connection (EVC) is added
- Notifies a CE device when an EVC is deleted
- Notifies a CE device of the current status of a configured EVC (that is, Active, Not Active, or Partially Active)
- Communicates via the Metro Ethernet Network (MEN) with User Network Interface (UNI) and EVC attributes to the CE device

ELMI works with most router architectures and is not specific to any hardware platform. It operates on both UNI-C and UNI-N to help facilitate communication between the two networks.

Figure 1-1 displays a simple diagram of an ELMI operation.

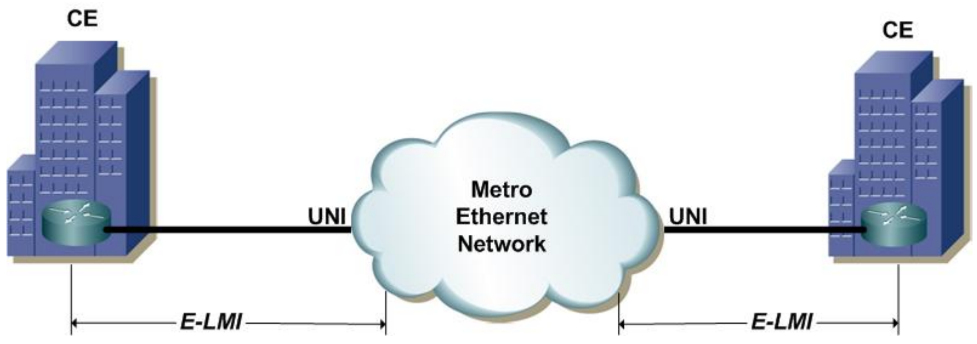

Figure 1-1: ELMI Operation

Features

The following are the features of the ELMI protocol:

- Full implementation of the MEF16 ELMI specification
- Includes both UNI-C and UNI-N functionality
- Has a local UNI significance between the MEN and the CE
- All-inclusive interaction among all new and existing EVC modules
- Includes a comprehensive set of APIs that can be managed through our purpose-built CLI

- Not designed for any type of hardware, so it is easy integrate with most types of routers and switches

Operation

The ELMI protocol enables a CE to request and receive status and service data from the MEN. This enables a CE device to configure its network to access Metro Ethernet services. The functionality of ELMI is similar to Frame Relay LMI. However, unlike FR-LMI, ELMI does not manage the link between the CE and the MEN, so users must provide link management between these two interfaces by some other means, such as the link management function in IEEE 802.3.

ELMI terminates by UNI-C on the CE side and by UNI-N on the MEN side. ELMI utilizes a point-to-point connection link to operate between the CE device and the PE (Provider Edge) device. In the ZebOS-XP architecture, all UNI functionality is located in NSM (Network Services Module). ELMI depends on the NSM for information regarding the attributes of both EVC and UNI. Furthermore, ELMI relies on CFM (Connectivity Fault Management) for end-to-end status of EVCs across the CFM domain.

Design

Each protocol module that supports ELMI uses ELMI client APIs to communicate with the ELMI base module. These APIs connect to and exchange information with the ELMI server module.

Software Integration

The ELMI module interacts with other modules, such as the NSM and other protocol modules, via the ELMI client. The ELMI client library in the protocol modules connects to the ELMI server library in the ELMI base module using a socket interface.

Management Interface

The ELMI module includes a command line interface (CLI). A management interface module is required to program ELMI-related parameters in the ELMI module. Typical operations are to set various timer values, set the ELMI mode, or to set the ELMI for a global or interface basis.

Application Protocol Interface

The ELMI Application Protocol Interface (API) provides the interface to the application protocols requiring forwarding plane liveliness detection.

CHAPTER 2 System Architecture

The following section describes the system architecture of the ELMI protocol.

System Implementation

To support the MEF-16 specification, ELMI is implemented as a separate daemon called “elmid.” During operation, ELMI opens a separate socket (AF_ELMI) with the kernel to send and receive control packets. Furthermore, the ELMI protocol operates on both UNI-C and UNI-N, with all UNI functionality located at the NSM. ELMI depends upon NSM for information regarding EVC and UNI attributes.

ELMI incorporates two client/server modules called the ELMI ONMD (OAM Network Management Daemon) client and the ELMI ONMD server, with the client operating on the ELMI protocol and the server operating on the ONMD module. ELMI uses socket-based messaging to interact with ONMD modules. The implementation of ELMI protocol includes the following components:

- NSM Client: For interaction with the NSM server
- ONM Client: For interaction with ONMD server to relay status of EVCs
- LACP ports: Support of ELMI on LACP (Link Aggregation Control Protocol) ports
- Base Module and Frame Handling: Support the ELMI initialization procedure and frame handling
- Configuration Manager: To interact with IMI and IMISH for ELMI protocol specific configuration

Enable the ELMI Protocol

ELMI is disabled by default, so it must be enabled it explicitly. Enable it on all devices by issuing the global enable CLI command, or enable it per interface by issuing the interface enable CLI command. If enabled globally, the ELMI protocol is also enabled on the bridge interfaces. Furthermore, ELMI is automatically enabled on any new port if a new port was added to the bridge after globally enabling of ELMI. Disable ELMI on a per interface basis.

System Operation

After enabling the ELMI protocol, it utilizes NSM for data regarding the attributes of both EVCs and UNIs. Upon receiving information from the NSM, the ELMI protocol running at UNI-N updates its data structure. It then sends all ELMI frames to UNI-C. Upon receiving this message, UNI-C buffers the received data and then updates its database with the latest information from UNI-N. UNI-C then sends a full status inquiry message when the polling timer starts. An operator can view the database structure by executing the related CLI commands. In addition, the NSM running on a PE device sends messages requesting updates related to the bridge, interface, and VLAN, as well as requests for status on EVC and attributes of UNI.

ELMI Framing Structure

The ELMI module’s frame structure is based on the IEEE 802.3 untagged MAC-frame format, and the ELMI module’s messages are encapsulated inside the Ethernet frames. By default, the destination address is 01-80-C2-00-00-07. The ELMI Ether type is 88-EE. The source address is the MAC address of the sending station or port.

Table 2-1 on page 12 displays the framing mechanism for the ELMI module:

Table 2-1: ELMI Framing Mechanism

Destination Address	Source Address	ELMI Ether-type	ELMI PDU (message)	CRC
6 Octets	6 Octets	2 Octets	46 - 1500 Octets (Data + Pad)	4 Octets

CE Auto-Configuration Support

The following are the supported CE auto-configuration features of the ELMI protocol:

- A CE-VLAN is created at UNI-C automatically if the CE bridge is configured for that CE-VLAN and if UNI-C receives information for any CE-VLAN in the CE-VLAN/EVC mapping, which does not exist at UNI-C. However, any existing (that is, manually configured) CE-VLAN configurations are not modified at UNI-C.
- Status information is updated in the database maintained at UNI-C. It receives a status for an EVC indicating that the EVC is Not Active. View the latest status of an EVC by issuing the appropriate show CLI commands.

Note: Currently, ELMI does not support the functionality of the CE to stop transmitting Service Frames on EVC until it receives the STATUS message indicating that EVC is Active or Partially Active.

- ELMI includes an API that triggers an alarm when EVC goes down at UNI-C.
- ELMI buffers information when UNI-C receives information for bandwidth profile attributes. View this information by issuing the appropriate show CLI command.

NSM Module

The following section describes an overview of the functionality of the NSM module.

How NSM Works

The following are the details on how the NSM works with the ELMI protocol:

1. NSM sends messages to ELMI with information about UNI status, EVC status, CE-VLAN to EVC mapping, and bandwidth profile attributes for UNI and EVC.
2. NSM sends UNI status information to the ELMI module, if the user has configured a port type as a CE port.
3. NSM sends the status of EVC when the user applies CVLAN-SVLAN registration table on the CE port. The status information for EVC also contains bitmaps for the CE-VLANS mapped to that EVC. The following APIs are invoked after the call:
   - nsm_vlan_send_port_map() [in functions]
   - nsm_cvlan_reg_tab_entry_add()
   - nsm_cvlan_reg_tab_if_apply()
   - nsm_pro_vlan_ce_port_sync()
4. NSM sends the delete EVC information to the ELMI. The following APIs are invoked:
   - nsm_cvlan_reg_tab_if_delete()
   - nsm_cvlan_reg_tab_handle_entry_delete()

5. NSM sends the following message to ELMI if a user modifies any bandwidth profile parameters at the UNI level:
   - NSM_MSG_ELMI_UNI_BW_ATTRIBUTES
6. NSM sends the following message to ELMI if the user configures any bandwidth profile parameters per EVC or per EVC per CoS level:
   - NSM_MSG_ELMI_EVC_BW_ATTRIBUTES
   - The bandwidth profile attributes are per EVC if no CoS bits are set in the parameter cos_id. Otherwise, the bits indicate the CoS ID for which the bandwidth profile attributes are applicable.
7. ELMI sends messages to the NSM to either indicate the ELMI operational status or to support auto-creation of CE-VLANS at UNI-C. The following messages are used:
   - NSM_MSG_ELMI_OPERATIONAL_STATUS
   - NSM_MSG_ELMI_CEVLAN_ADD_TO_UNIC
8. ELMI sends a status message to NSM once it detects that ELMI is operational (N391-1) or detects that there is a change in the operational status of ELMI if a user enables ELMI on a UNI.

Note: To send an ELMI operational status at UNI-N from the ELMI module to the NSM, a polling verification timer is enabled at UNI-N. The timer never expires and the operational status of the ELMI cannot be determined at UNI-N if the user disables the polling verification timer.

NSM Client Parser and Callback

The NSM client installs parsers for each message received from the NSM. In addition, the ELMI module registers a callback function that is called by the respective parser routine. The prototype of the callback function is as follows:

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

Where:

- header: Pointer to the NSM message header
- arg: Pointer to the NSM connection handler
- message: Pointer to the message structure

The function *nsm_client_create* is updated with the parser functions required for communication between the NSM and ELMI. Any new parser function is added to any new message introduced between the NSM and the ELMI protocol.

NSM Client Disconnect Handling

If the ELMI-NSM client installs a disconnect handler, the library for the ELMI client calls a disconnect handler when a disconnect occurs. When the disconnect handler is registered, a reconnection is initiated if a loss of connection to NSM occurs.

ONMD Module

The ONMD module sends the status information for EVCs to the ELMI module. Communication between the two modules occurs when a new socket opens and messaging is used. An ONM client module is attached to the ELMI protocol to help facilitate the communication between the modules. A server module also runs in parallel on ONMD.

ELMI ONM client registers for the following services with the ONM:

- ONM_MSG_SERVICE_EVC

ONM registers for the following ONM messages:

- ONM_MSG_EVC_STATUS

The ONM client in ELMI connects to the ONM server using a socket interface. At system startup, ELMI creates the ONM client and then starts the module. ONM client is notified from ONMD for status changes of EVC, which in turn sends the notification to the ELMI protocol. Each message sent from ONMD server to ELMI contains the following information:

- Message Type
- Message Length
- Message Data in the TLV encoded (length/field) format

The API `onm_client_start` starts the ONM client and establishes a TCP/Unix domain socket connection with the ONMD server, with a reconnection interval set. In addition, the TCP port 4800 is used by the ONM server to listen for connection requests from the ONM client. The path `/tmp/.onmserv` is used if Unix domain sockets are used for ONM client and ONM server communication. The ONM_PORT is used for TCP sockets.

The ONM client is started during ELMI protocol daemon startup, and initiates the connection with the ONM server. If the ONM server is not running, the ONM client retries the connection in five-second intervals. ELMI utilizes CFM (Connectivity Fault Management) for the status of EVCs, so the ONMD module is required to communicate with the EVCs for information coming from the ELMI module. The communication is sockets based.

CE-VLANs

The following section describes how to create and delete CE-VLAN (Virtual Local Area Networks).

Auto-creation of CE-VLAN

To handle the auto-creation of CE-VLAN at UNI-C, ELMI sends a message to NSM to add the CVLAN to UNI-C. An auto-configured CVLAN is removed from UNI-C if it receives information from UNI-N that the CE-VLAN is not mapped to an EVC.

A CE-VLAN is created at UNI-C if it receives information from UNI-N about mappings of CE-VLAN and EVC for a CE-VLAN that does not exist at UNI-C. To support this functionality, ELMI sends a message to NSM to add the CVLAN.

The following is the process for how the ELMI handles to auto-creation of CE-VLANs:

1. The ELMI receives date about the creation of a CE-VLAN
2. A check is done at ELMI to see whether the CVLAN is present in the CVLAN bitmap.
3. A message is sent to NSM to add this CE-VLAN if it is not present.
4. When the NSM receives the message, it checks whether the CE-VLAN is already configured on the CE bridge.

5. ELMI invokes the API to add the CE-VLAN to a bridge if the CE-VLAN is not configured on the CE bridge:
    - nsm_vlan_add()
6. If successful, the following API is called to configure the CE-VLAN on UNI-CAPI:
    - nsm_add_vlan_to_port()
7. An error message is logged at NSM if it is still not able to add the CE-VLAN to bridge. NSM also sends a reply back to ELMI about the success or failure of the CVLAN addition or deletion.
8. ELMI waits for the reply message from NSM by calling the following API:
    - nsm_client_read_sync()
9. The following API is called (without calling the "nsm_vlan-add()" API) to add the CE-VLAN to UNI-C if the CE-VLAN is already configured on the CE bridge:
    - nsm_add_vlan_to_port()
10. To ensure that user-created VLANs and dynamic VLANs are not deleted through the ELMI at the NSM, the following separate flags are maintained for each CE-VLAN type:
    - NSM_VLAN_ELMI_AUTO
    - NSM_VLAN_STATIC
    - NSM_VLAN_DYNAMIC

Auto-configured CE-VLAN Deletion

An auto-configured CE-VLAN can be deleted using the CLI. This ensures that no "zombie" CE-VLANs are left in the CE bridge if, for some reason, the ELMI becomes non-operational. If UNI-C receives information from UNI-N regarding deletion of an auto-configured CE-VLAN, ELMI sends a message to NSM to delete the CE-VLAN. At the NSM, UNI-C is removed from the CE-VLAN and a check is done to see if any other bridge port is also part of the CE-VLAN. The CE-VLAN is removed from the bridge if no other bridge port is part of the CE-VLAN.

If CE-VLANs are deleted that were derived from ELMI, those CE-VLANs are added again by ELMI once it receives the next full status message from UNI-N, assuming that the mapping has not changed at UNI-N. The following APIs are called to send the VLAN add/delete requests from ELMI to NSM:

- nsm_msg_elmi_auto_vlan_add ();
- nsm_msg_elmi_auto_vlan_delete();

The following APIs are callback functions that are implemented in NSM to receive these messages from ELMI:

- nsm_server_recv_elmi_auto_vlan_add ();
- nsm_server_recv_elmi_auto_vlan_delete ();

CHAPTER 3
System Messaging

This section provides a general overview of system messaging for the ELMI module. For additional details on system messaging. See See Appendix A, *System Messaging Details*.

System Timers

A system timer generates recurring events and messages in an application. The ELMI module supports the following system timers.

Polling Timer

UNI-C sends a status message request to UNI-N at a specified interval. The range is from 5 through 30 seconds, with a default value of 10 seconds. During system startup or restart, ELMI initializes the polling counter to the default Polling Counter value (360), Polling Time to the default interval (10 seconds) and the Data Instance field to zero. In addition, the Periodic Polling timer starts up on the ELMI module.

Polling Timer Processes

As soon as the polling timer starts, UNI-C checks if the polling counter is zero. If it is not zero, UNI-C decrements the polling counter and sends a status inquiry message (that is, "ELMI Check") with the local Data Instance (DI) included. On receipt of this message, UNI-C compares the received UNI-N DI with its local ID and takes the following actions:

- A mismatch indicates that UNI-N has new data to send and UNI-C immediately sends a full-status inquiry message with its local DI value. No action is taken if they are equal.
- UNI-C parses the status message depending on the report type. If it is a full status, or full status continued status message, UNI-C updates its configuration according to the status of UNI and the status and service attributes of each configured EVC.
- Operational status of ELMI is declared as non-operational if the most recent N393 polling timer expires without any responses from any UNI-N.
- Status of ELMI is changed to operational only if the recent N393 timer expires are successful and without any errors if ELMI was not operational at the previous polling timer expiry.
- To distinguish between the far-end ELMI "non-operational" and far-end "restart" states, a receive sequence number is used. If the receive sequence number is zero, this indicates that the other end has started or restarted.
- For UNI-C, if N393 is three, it receives a response from UNI-N due to an ELMI restart at UNI-N. In addition, UNI-C check is down if the receive sequence number is zero. N393 is set to zero and declares ELMI as operational if it is at zero. Otherwise, the ELMI is declared operational only if there are no errors in the subsequent N393 consecutive Polling Timer expirations.
- For UNI-C, the entire configuration is deleted, including EVCs/UNIs and CE-VLAN mapping, if ELMI detects that it is not operational. However, configurations not created at UNI-N using ELMI are not deleted.

Polling Verification Timer

The Polling Verification Timer (PVT) specifies the interval for receiving status inquiry messages in seconds. This timer records errors before the timer expires if no messages are received. The range is from 5 through 30 seconds, with a default value of 15 seconds. The PVT value must always be greater than the Polling Timer.

Enable or disable the polling verification timer using the CLI. Starting or restarting the PVT resets the timer. The PVT is enabled by default and is optional at UNI-N.

Polling Verification Timer Processes

The following are the various processes for PVT, including the start, restart and reset procedures for the timer:

- When UNI-N receives a status inquiry message from UNI-C, it replies with a status message and resets the Polling Verification Timer.
- PVT resets if a value is configured other than zero for t392
- On receipt of status inquiry message, a full status inquiry message, or a full status continued inquiry message from UNI-C, UNI-N responds to the request and restarts PVT.
- UNI-N logs an error and increments the status counter if it does not receive any request from UNI-C before the expiry of the PVT.
- UNI-N declares the ELMI module running at UNI-C as not operational if the status counter value reaches the configured threshold (that is, the maximum number of consecutive errors).
- The operational state the ELMI module changes to operational and a notification is sent to NSM if UNI-N receives consecutive N393 status inquiry messages from UNI-C without expiry of PVT.

Full Status Polling Interval

For the full status polling interval, UNI-C sends a status inquiry to UNI-N at the interval specified by the n391. The N391 specifies the frequency with which these inquiries expect a full status report. For example, an n391 value of “360”0 specifies a full status report in response to every 360th inquiry. The intermediate inquiries ask for a keep-alive exchange, only. The range is 1 through 65k, with a default value of 10.

Status Counter

The status counter keeps the count of consecutive errors. The range is 1 through 10, with a default value of 4.

System Reporting

The ELMI module supports the following report types:

- Full Status
- Full Status Continued
- EVC Asynchronous Status
- ELMI Check

Full Status

The following section describes the details of the full status messages.

UNI-C

During system initialization, UNI-C sets the polling counter to the default value of 360. On every polling timer expiry, this counter decrements by one. UNI-C sends a full status inquiry message if the counter reaches zero and the polling timer expires. UNI-C checks if the polling counter is zero when the polling timer expires. UNI-C sends a full status inquiry message to UNI-N if it is zero.

UNI-N responds with a status message containing status information about the EVC elements for each EVC at UNI when UNI-C sends a status inquiry message with a report type of full status. Each EVC information element contains an active bit and a partially active bit indicating the availability or unavailability of an EVC. UNI-N sends an EVC information element with the “New” bit set to 1 and the status of EVC equal to Active, Not Active, or Partially Active.

Full Status Continued

When the information for all EVCs cannot be supported by the maximum Ethernet frame size, UNI-N sends information regarding EVCs using multiple full status continued messages. When UNI-C receives a status message with the report type of “full status continued,” UNI-C transmits a status inquiry with a report type of full status continued to obtain status and service attributes for additional EVCs.

Asynchronous Status

The Asynchronous Status messages are status flags that check to control the written register updates. ELMI includes the following Asynchronous Status messages.

UNI-N

The function of the UNI-N asynchronous status message is to notify UNI-C when EVC has changed status, without having to wait for a request from UNI-C. Whenever there is a change in the status of an EVC, UNI-N sends an EVC Asynchronous Status message to UNI-C.

Note: The interval between asynchronous messages is greater than or equal to 1/10th of T391.

The following is the sequence of operations involved in an EVC asynchronous status message:

1. NSM sends a message to ELMI if the user creates a CE-VLAN to EVC mapping on UNI-N. In addition, UNI-N updates its local DI and adds this DI to the next ELMI check or full status messages sent to UNI-C.
2. UNI-C sends a full status inquiry message to UNI-N upon detecting a mismatch in its local DI with the DI received in the status message.
3. UNI-N sends a full status message to UNI-N upon receiving a full status inquiry message.
4. CFM monitors the status of EVCs through CCM.
5. ONMD sends a notification message to ELMI if CFM detects that an EVC link is down.
6. One of the following two events occur if ELMI running at UNI-N notices that the interval between two successive asynchronous messages is less than or greater than 1/10th of Polling Timer (T391):
    - If it is greater, ELMI sends an EVC asynchronous status message to UNI-C.
    - If it is less, UNI-N does not send a message containing any UNI-C information. The message only contains the information of a single EVC.
7. UNI-C updates its database if it receives an asynchronous status message for an EVC from UNI-N.
8. The latest status of EVCs reflects in the output of the appropriate show CLI command.

Data Instance Triggered Update

In addition to triggering full status and full status continued reports every N391 polling cycles, the Data Instance (DI) is used to trigger these reports each time there is a change in EVC or UNI information. This eliminates the delay in receiving this information that UNI-C would experience if the information was only sent every N391 polling cycles.

UNI-C

When the UNI-C starts or restarts, it sets its DI to zero (0). UNI-C sends an ELMI status inquiry with the report type of "full status." UNI-C then receives full status or full status continued reports, including the latest information on both UNI and EVC that was stored in the local database. Afterwards, the local DI for UNI-C sets to UNI-N DI, which it received from the full status report. For the ELMI full status continued report, the DI value sent by UNI-N does not change until the status procedure is complete. Any full status procedure with errors results in UNI-C not updating the local DI to UNI-N DI value and a full status inquiry triggers.

UNI-N

When UNI-N first comes up, it sets its DI value to a non-zero value that is different from the DI value received in the first message received from UNI-C. Any change in information related to either UNI or EVC, including status change, results in incrementing DI value to reflect the change in data. The value zero is skipped whenever UNI-N changes to increment the DI. For full status and full status continued reports, the DI value sent by UNI-N does not change from what was sent in the first full status continued report until after the status procedure is complete. For the ELMI check, upon receipt of the ELMI status inquiry with the ELMI check, UNI-N responds with the status of ELMI and includes the current value of DI.

Sequence Numbers

The purpose of the Sequence Numbers information element is to allow UNI-N and UNI-C to determine the status of the ELMI process including correlating status inquiry messages with status messages. Both UNI-C and UNI-N maintains a send sequence counter and receive sequence counter. Before messages are exchanged, UNI-C and UNI-N set the send sequence counter and receive sequence counter to zero. The sequence number is calculated based on modulo 256. If the sequence number value exceeds the six (6) bit counter, it rolls over to zero; thus, the sequence number starts from zero once it reaches the maximum value of 64.

UNI-N

Whenever UNI-N receives a status inquiry from UNI-C, UNI-N checks the "receive" sequence number received from UNI-C against its "send" sequence counter. If the values do not match, an error occurs and the received send sequence number is stored in the receive a sequence counter. UNI-N then increments its send sequence counter and places its value in the send sequence number field. The value of the receive sequence counter (that is, the last received send sequence number) goes into the receive sequence number field of the outgoing sequence numbers information element. UNI-N then transmits the completed status message back to UNI-C. UNI-N increments the send sequence counter using modulo 256 and skips the value zero.

UNI-C

Before any messages are exchanged, UNI-N sets the "send" sequence counter and "receive" sequence counters to zero. Each time UNI-C sends a status inquiry message, it increments the send sequence counter and places its value into the send sequence number field. It also place the current value of the receive sequence counter into the receive sequence number field of the Sequence Numbers information element. UNI-C increments the send sequence counter using modulo 256 and skips the value zero.

When UNI-C receives a status message from UNI-N in response to a status inquiry, UNI-C checks the receive sequence number received from UNI-N against its send sequence counter. If the values do not match, an error occurs. The received send sequence number stores in the receive sequence counter.

Reporting a New EVC

When a new EVC is added, UNI-N increments its DI and sets the new bit to one (1) for the EVC status element in the next full status or full status continued status message sent. The “New” bit in the EVC status data element sets to one (1) until the DI in the status inquiry received by UNI-N is equal to the DI sent by UNI-N in the message that the new bit was set to one (1). UNI-C does one of the following when it receives a full status or full status continued status message containing an EVC information element with the New bit of the EVC status information element set to one (1):

- UNI-C deletes EVC from its list and adds the new EVC to its list of configured EVCs if it has an EVC with the same EVC reference ID in its list of configured EVCs.
- UNI-C does not update its local DI value if it is not able to update the new EVC information in its database.
- UNI-N sets the New bit of the EVC status information element to zero (0) when an existing EVC attribute (for example, Bandwidth profile) changes.

Error Messages

The following error conditions are handled by both UNI-N and UNI-C:

- Reliability errors: these errors include non-receipt of status/status inquiry messages or invalid sequence numbers in a sequence numbers information element.
- Protocol errors: UNI-N and UNI-C ignore messages (including their sequence numbers) containing protocol errors.
- UNI-C and UNI-N ignore unrecognized information and sub-information elements. No counters are supported by the ELMI module for these types of errors.

CHAPTER 4 Data Structures

This chapter describes the data structures that are used with the ELMI module.

Common Data Structures

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

- `cli`
- `interface`

elmi_ifp

This data structure is defined in the `elmid/elmi_types.h` file.

Definition

```
struct elmi_ifp
{
  struct interface *ifp;
  struct elmi_bridge *br;
  struct elmi_master *elmim;
  struct elmi_port *port;
  struct elmi_vlan_bmp *vlan_bmp;
  struct list *evc_status_list; ///<List of elmi_evc_status nodes
  struct elmi_uni_info *uni_info;
  struct list *cevlan_evc_map_list; ///<List of CE-vlan to EVC mapping node
  u_int32_t ifindex;
  u_int16_t pvid;

  bool_t active;
  u_int8_t dev_addr [ELMI_MAC_ADDR_LEN];
  u_int8_t elmi_enabled; ///< Flag to indicate elmi enable/disable state
  u_int8_t uni_mode;   ///< Flag to indicate UNI-C/UNI-N mode
  u_int8_t port_type;
  u_int8_t bw_profile_level; ///< Flag to indicate BW profile type
                           ///(Per UNI/EVC/EVCCOS)

  /* Field to maintain configured polling time interval */
  u_int8_t  polling_time;

  /* Field to maintain configured polling verification time interval */
  u_int8_t  polling_verification_time;
  /* Field to maintain configured status counter threshold value */
  s_int8_t  status_counter_limit;

  /* flag to disable PVT */
  u_int8_t  enable_pvt;

  /* Field to maintain configured polling counter threshold value */
  u_int16_t polling_counter_limit;

  /* Field to maintain Single EVC Asysnchronous Timer */
  u_int8_t  asynchronous_time;

};
```

nsm_msg_header

This data structure is defined in the `elmid/elmi_types.h` file.

Definition

```
struct nsm_msg_header
{
  /* VR-ID. */
  u_int32_t vr_id;

  /* VRF-ID. */
  u_int32_t vrf_id;

  /* Message Type. */
  u_int16_t type;

  /* Message Len. */
  u_int16_t length;

  /* Message ID. */
  u_int32_t message_id;
};
```

CHAPTER 5 ELMI Command API

The functions defined in this chapter are called by the commands described in the *Ethernet Local Management Interface Command Reference*.

elmi_enable_global

This API is called to enable ELMI on all the interfaces configured on a bridge. ELMI is enabled only on interfaces that are part of a bridge. After enabling ELMI at the global level, ELMI is enabled on those ports if they are added to the bridge. An elmi_enabled flag sets in the bridge data structure when this API is executed.

CLI Command

```
    ethernet lmi global
```

API Call

```
s_int32_t
elmi_api_enable_bridge_global (u_char *br_name)
```

Input Parameters

`br_name`	Bridge on which the ELMI is to be created.

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_disable_global

This API is called to disable ELMI on all the interfaces configured on a bridge. The elmi_enabled flag resets to ELMI_FALSE in the bridge data structure when this API is executed.

CLI Command

```
    no ethernet lmi global
```

API Call

```
s_int32_t
elmi_api_disable_bridge_global (u_char *bridge_name)
```

Input Parameters

`bridge_name`	Bridge on which the ELMI is to be created.

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_enable_port

This API is used to enable the ELMI protocol on a port.

CLI Command

```
    ethernet lmi interface
```

API Call

```
s_int32_t
elmi_api_enable_port (u_int32_t ifindex)
```

Input Parameters

`ifindex`	Interface index

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_disable_port

This API is used to disable ELMI on a port.

CLI Command

```
    no ethernet lmi interface
```

API Call

```
s_int32_t
elmi_api_disable_port (u_int32_t ifindex)
```

Input Parameters

`ifindex`	Interface index

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_set_port_polling_time

This API is used to set polling time interval for the polling timer to expire.

CLI Command

```
    ethernet lmi [n391 n393 t391 t392] value
```

API Call

```
s_int32_t
elmi_api_set_port_polling_time (u_int8_t *ifName,
                                u_int8_t polling_time)
```

Input Parameters

`ifname`	Interface name
`polling_time`	Polling time

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_set_port_polling_verification_time

This API is used to set polling verification time interval for polling verification timer to expire.

CLI Command

```
    ethernet lmi [n391] value
```

API Call

```
s_int32_t
elmi_api_set_port_polling_verification_time (u_int8_t *ifName,
                                             u_int8_t polling_verification_time)
```

Input Parameters

`ifname`	Interface name
`polling_verification_time`	Polling verification time

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_set_port_polling_counter

This API is used to configure polling counter at UNI-C.

CLI Command

```
    ethernet lmi [n393] value
```

API Call

```
s_int32_t
elmi_api_set_port_polling_counter (u_int8_t *ifName,
                                   u_int16_t polling_counter)
```

Input Parameters

`ifname`	Interface name
`polling_counter`	Polling counter

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_api_set_port_status_counter

This API is used to configure status counter at both UNI-C and UNI-N.

CLI Command

```
    ethernet lmi [t391] value
```

API Call

```
s_int32_t
elmi_api_set_port_status_counter (u_int8_t *ifName,
                                  s_int8_t status_counter)
```

Input Parameters

`ifName`	Interface name
`status_counter`	Status counter

Output Parameters

None

Return Values

RESULT_OK for success

RESULT_ERROR for failure

elmi_clear_statistics_interface

This API is used to clear the ELMI statistics information per interface.

CLI Command

```
    clear ethernet lmi statistics interface IFNAME
```

API Call

```
int
elmi_clear_statistics_interface (struct cli *cli, struct elmi_ifp *elmi_if);
```

Input Parameters

None

Output Parameters

None

elmi_clear_statistics_bridge

This API is used to clear the ELMI statistics of all bridge interfaces.

CLI Command

```
    clear ethernet lmi statistics [bridge <id>]
```

API Call

```
int
elmi_clear_statistics_bridge (struct cli *cli, u_int8_t *bridge_name);
```

Input Parameters

`bridge_name`	Bridge on which the ELMI is to be created

Output Parameters

None

CHAPTER 6 ELMI Module API

The following are the functions included with the ELMI module.

ELMI Callbacks/Handlers for NSM Messages

This subsection includes the ELMI Callbacks/Handlers for NSM Messages.

elmi_nsm_recv_service

This callback processes the service message received from the NSM.

API Call

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

elmi_interface_add

These callbacks handle any interface addition or deletion.

API Call

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

Input Parameters

`ifp`	Interface structure.

Output Parameters

None

elmi_nsm_recv_interface_state

Calls the appropriate `enable_port` or `disable_port` command API, based on whether the link message received is `NSM_MSG_LINK_UP` or `NSM_MSG_LINK_DOWN`, respectively.

API Calls

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

Input Parameters

`header`	The message header.
`arg`	The argument for the message.
`message`	The message to be handled.

Output Parameters

None

Return Values

`RESULT_ERROR` if port for which message was received is not found, else,

`RESULT_OK`

elmi_nsm_recv_bridge_add

Calls the Bridge Add message from NSM.

API Call

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

Return Values

Always 0 (zero)

elmi_nsm_recv_bridge_delete

Calls the bridge delete message from NSM.

API Call

```
static int
elmi_nsm_recv_bridge_delete (struct nsm_msg_header *header,
                             void *arg, void *message);
```

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

Return Values

Always 0 (zero)

elmi_nsm_recv_bridge_add_if

Adds one or more ports to the bridge by calling the appropriate `bridge_add_port` command API.

API Call

```
static int
elmi_nsm_recv_bridge_add_if (struct nsm_msg_header *header,
                             void *arg, void *message);
```

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

Return Values

Always 0 (zero)

elmi_nsm_recv_bridge_delete_if

Deletes one or more ports from the bridge by calling the appropriate `bridge_delete_port` command API.

API Call

```
static int
elmi_nsm_recv_bridge_delete_if (struct nsm_msg_header *header,
                                void *arg, void *message);
```

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

Return Values

Always 0 (zero)

elmi_nsm_recv_vlan_port_type

Processes the change of port type message from NSM. If it is a customer edge port, it creates a spanning tree domain for the customer edge port.

API Call

```
static int
elmi_nsm_recv_vlan_port_type (struct nsm_msg_header *header,
                          void *arg, void *message);
```

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

Return Values

Always RESULT_OK

elmi_nsm_recv_msg_done

API Call

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

nsm_evc_send_evc_add_msg

The following are the handlers and callbacks that send UNI and the EVC attributes from the NSM to the ELM.

API Call

```
nsm_evc_send_evc_add_msg (struct nsm_msg_elmi_evc *msg,
                          int msgid)
```

Input Parameters

`msg`	Pointer to the message structure.
`msgid`	Message identifier.

Output Parameters

None

elmi_nsm_recv_evc_add

Decodes the message attribute, updates the data structure, and updates data instance field.

API Call

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

elmi_nsm_recv_evc_delete

This API triggers when the user deletes SVLAN-CVLAN registration table mapping from UNI-N.

API Call

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

Input Parameters

`bridge`	Bridge on which EVC should be deleted.
`msg`	Pointer to the message structure.

Output Parameters

None

elmi_nsm_recv_evc_delete

Decodes the message attribute, updates the data structure, updates the data instance field, and sends notification to UNI-C.

API Call

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

nsm_server_send_evc_update

Check for service type, encodes the fields, and invokes the NSM server's send message.

API Call

```
s_int32_t
nsm_server_send_evc_update (struct nsm_server_entry *nse,
                            struct interface *ifp, u_int16_t evc_id,
                            struct nsm_svlan_reg_info *svlan_info,
                            cindex_t cindex)
```

Input Parameters

`*nse`	NSM server entry
`ifp`	Reference to interface structure
`evc_id`	EVC identifier.

Output Parameters

`svlan_info`	Service VLAN info.

Output Parameters

None

elmi_nsm_recv_evc_update

Decodes the message attribute, updates the local data structure, and updates the data instance field.

ELMI API

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

Input Parameters

`header`	Pointer to the NSM message header.
`arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

nsm_server_uni_update

Encodes the message attributes and invokes the NSM server send message.

API Call

```
void
nsm_server_uni_update (struct nsm_bridge_port *br_port, struct interface *ifp,
                       u_int32_t cindex)
```

Input Parameters

`*br_port`	Bridge port of the EVC.
`*ifp`	Reference to interface structure

Output Parameters

None

elmi_nsm_recv_uni_update

Decodes the message attributes, updates the local data structures, and updates the data instance field.

API Call

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

Input Parameters

`*header`	Pointer to the NSM message header.
`*arg`	Pointer to the NSM connection handler.
`message`	Pointer to the message structure.

Output Parameters

None

nsm_elmi_send_uni_add

Encodes the message and invokes the NSM server's send message.

API Call

```
s_int32_t
nsm_elmi_send_uni_add (struct nsm_bridge * bridge, struct interface *ifp)
```

Input Parameters

`bridge`	Bridge of the EVC.
`*ifp`	Reference to interface structure

Output Parameters

None

nsm_elmi_send_evc_add

Checks for the service type, encodes the fields, and invokes the NSM server's send message.

API Call

```
s_int32_t
nsm_elmi_send_evc_add (struct nsm_bridge *bridge,
                       struct interface *ifp,
                       u_int16_t svid,
                       struct nsm_svlan_reg_info *svlan_info)
```

Input Parameters

`bridge`	Bridge of the EVC.
`msg`	Pointer to the message structure.
`msgid`	Message identifier.

Output Parameters

None

ONM APIs

The following APIs are implemented for interaction between ONM server and ONM client.

onm_client_create

This API is used to create ONM client.

API

```
struct onm_client *
onm_client_create (struct lib_globals *zg, u_int32_t debug)
```

Input Parameters

`cb`	Name of the callback.
`zg`	Name of the bridge.

Output Parameters

None

onm_client_read_msg

This API is used to read message from ONM sever.

API

```
int
onm_client_read_msg (struct message_handler *mc,
                     struct message_entry *me,
                     pal_sock_handle_t sock);
```

Input Parameters

`mc`	Pointer to the NSM multicast structure.
`me`	Pointer to the NSM multicast-enabled structure.

Output Parameters

None

onm_client_connect

This API is used to establish connection with ONM server.

API

```
int
onm_client_connect (struct message_handler *mc, struct message_entry *me,
                    pal_sock_handle_t sock)
```

Input Parameters

`mc`	Pointer to the NSM multicast structure.
`me`	Pointer to the NSM multicast-enabled structure.

Output Parameters

None

onm_client_reconnect_start

This API is used to reconnect to ONM server.

API

```
int
onm_client_reconnect_start (struct onm_client *oc);
```

Input Parameters

`oc`	Name of the ONM client.

Output Parameters

None

onm_client_start

This API is used to start ONM client.

API

```
int
onm_client_start (struct onm_client *oc)
```

Input Parameters

`oc`	Name of the ONM client.

Output Parameters

None

onm_client_send_connect

This API is used send connect message to ONM server.

API

```
int
onm_client_send_service (struct onm_client_handler *och)
```

Input Parameters

`och`	Name of the ONM client handler.

Output Parameters

None

onm_client_send_message

This API is used to send messages.

API

```
int
onm_client_send_message (struct onm_client_handler *och, u_int32_t vr_id,
                         int type, u_int16_t len, u_int32_t *msg_id)
```

Input Parameters

`cc`	Name of the client.

Output Parameters

None

onm_client_reconnect

The functionalities handled by the APIs onm_client_reconnect_start and onm_client_reconnect are similar to the nsm_client_reconnect () and nsm_client_reconnect_start() APIs. The API onm_client_reconnect_start () is called by onm_client_disconnect () API.

If the client registers for any disconnect_handler, the disconnect function tries to reinitiate a connection to ONM server using a reconnect timer thread if there is a loss of connection. The function onm_client_reconnect () is a callback function to the reconnect timer thread. The retry interval is 5 seconds.

API

```
int
onm_client_reconnect (struct thread *t)
```

Input Parameters

`t`	Name of the thread.

Output Parameters

None

ELMI Init Module and Frame Handling APIs

This subsection includes the ELMI init module and frame handling APIs.

elmi_start

As soon as the ELMI daemon starts, the function elmi_start executes, which handles the protocol initialization.

```
s_int32_t
elmi_start (s_int32_t deamon_mode, char *config_file,
            s_int32_t vty_port, char *progname)
```

Input Parameters

`config_file`	Name of the config file.
`progname`	Program name.

Output Parameters

None

elmi_enable_port

This AP invokes whenever a user enables ELMI on a port.

API

```
s_int32_t
elmi_enable_port (struct elmi_bridge *br, struct interface *ifp);
```

Input Parameter

`ifp`	Reference to interface structure

Output Parameter

None

elmi_unic_tx

This function sends different types of frames from UNI-C. It is invoked whenever the timer expires. It is a common function to send status inquiry messages and full statues messages, which in turn calls elmi_frame_send to send frames to layer2 module.

API

```
int
elmi_unic_tx(struct elmi_port *port, u_int8_t frame_type)
```

Input Parameter

`port`	Port on which the ELMI is added.

Output Parameter

None

ELMI Frames

The following APIs handle the transmission and reception of ELMI frames in the control plane.

elmi_sock_init

API

```
pal_sock_handle_t
elmi_sock_init (struct lib_globals *zg)
```

Input Parameter

None

Output Parameter

None

elmi_frame_recv

API

```
s_int32_t
elmi_frame_recv (struct thread *thread)
```

Input Parameter

None

Output Parameter

None

elmi_frame_send

API

```
s_int32_t
elmi_frame_send (u_char *src_addr, const u_char *dest_addr,
                 u_int32_t ifindex , u_char *data, u_int32_t length)
```

Input Parameter

None

Output Parameter

None

Appendix A System Messaging Details

The following section describes the details of system messaging functionality for the ELMI protocol.

Message Types

The following are the two types of messages defined for the ELMI protocol:

- STATUS and
- STATUS ENQUIRY

Message Fields

The ELMI protocol includes the following fields that are displayed in all system messages:

- Protocol Version
- Message Type
- Report Type
- Sequence Numbers (Not included for Asynchronous EVC Status message)
- Data Instance (Not included for Asynchronous EVC Status message)
- UNI Status: Included in the full status report to indicate the status and parameters of UNI
- EVC Status: Included in the full status report to indicate full status, the status of an EVC asynchronous state, and if UNI has EVCs configured.
- CE-VLAN ID/EVC Map: Included in the full status report.

System Behavior

Defined by the procedure, the system behavior of ELMI triggers a message to be carried out by the module. These procedures are characterized by the messages that are exchanged at UNI. The behavior includes the following:

- The events at the CE and the MEN
- The ELMI module’s messages received by UNI-C and UNI-N

Protocol Relays

The ELMI module relays the following information from UNI-N to UNI-C:

- Notification to the CE of the addition/deletion of an EVC (active, not active, or partially active)
- Notification to the CE of the availability state of a configured EVC
- Communication of UNI and EVC attributes to the CE

UNI Attributes

The following UNI attributes display in system message:

- UNI Identifier (a user-configured name for UNI)
- CE-VLAN ID/EVC map type (all-to-one bundling, service multiplexing with bundling or no bundling)

The following bandwidth profile attributes display in system message for UNI:

- CM (coupling mode)
- CF (color flag)
- CIR (committed Information rate)
- CBR (committed burst size)
- EIR (excess information rate)
- EBS (excess burst size)

EVC Attributes

The following EVC attributes display in system message:

- EVC reference ID
- EVC status type (active, not active, or partially active)
- EVC type (point-to-point or multipoint-to-multipoint)
- EVC ID (a user-configured name for EVC)
- CE-VLAN ID/EVC map

The following bandwidth profile attributes display in system message for EVC:

- CM (coupling mode)
- CF (color flag)
- CIR (committed Information rate)
- CBR (committed burst size)
- EIR (excess information rate)
- EBS (excess burst size

Receiving a Message

On receipt of a status messages from UNI-N to UNI-C, the frame is parsed at UNI-C and the contents is buffered in the database maintained at UNI-C.

NSM Messaging

NSM sends a message to the ELMI protocol to receive information on the status of UNI, EVC, mapping of CE-VLAN/EVC and bandwidth profile attributes for UNI and EVC. In addition, NSM sends a UNI status message to ELMI whenever a user configures a port type as a CE port. It also sends bandwidth profile parameters for this port. SM sends EVC status information whenever a user applies CVLAN-SVLAN registration table on a CE port. The status information also contains bitmaps for the CE-VLANs mapped to that EVC.

The following APIs are invoked after this call:

- nsm_vlan_send_port_map() [in functions]
- nsm_cvlan_reg_tab_entry_add()
- nsm_cvlan_reg_tab_if_apply()
- nsm_pro_vlan_ce_port_sync()

NSM sends EVC delete information messaging to the ELMI module. The following APIs are invoked after this call:

- nsm_cvlan_reg_tab_if_delete()
- nsm_cvlan_reg_tab_handle_entry_delete().

NSM sends a message to ELMI with the following message if any bandwidth-profile parameter is modified at UNI level:

- nsm_msg_elmi_uni_bw_attributes

NSM sends that information to ELMI using the following message type if any bandwidth profile parameters were configured per EVC or per EVC per cos-level:

- nsm_msg_elmi_evc_bw_attributes

NSM indicates that the bandwidth profile attributes are per EVC if no CoS bits are set in the parameter cos_id. Otherwise, the bits indicate the CoS ID for which the bandwidth profile attributes are applicable.

ELMI sends messages to NSM to indicate the ELMI operational status and to support auto-configuration of the CE-VLANs at UNI-C. The following messages are used:

- NSM_MSG_ELMI_OPERATIONAL_STATUS
- NSM_MSG_ELMI_CEVLAN_ADD_TO_UNIC

ELMI sends a status message to NSM once it detects that ELMI is operation if this protocol is enabled on a UNI. In addition, ELMI sends notifications to NSM whenever there is a change in the operational status of ELMI. To send the operational status at UNI-N from ELMI to NSM, enable the polling verification timer at UNI-N. If the polling verification timer is disabled, the timer never expires and the operational status of ELMI cannot be determined at UNI-N. The following services are provided by the NSM to the ELMI protocol. The ELMI NSM client registers with NSM on startup with the following service bits:

- NSM_SERVICE_INTERFACE
- NSM_SERVICE_BRIDGE
- NSM_SERVICE_VLAN
- NSM_SERVICE_EVC
- NSM_MSG_SERVICE_REPLY

ELMI-NSM-Client Messaging

The ELMI-NSM-Client module registers for the following messages with NSM:

Link Messages

- NSM_MSG_LINK_UP
- NSM_MSG_LINK_DOWN

Bridge/VLAN Messages

- NSM_MSG_BRIDGE_ADD
- NSM_MSG_BRIDGE_DELETE

- NSM_MSG_BRIDGE_ADD_PORT
- NSM_MSG_BRIDGE_DELETE_PORT
- NSM_MSG_VLAN_PORT_TYPE
- NSM_MSG_VLAN_ADD
- NSM_MSG_VLAN_DELETE
- NSM_MSG_VLAN_ADD_PORT
- NSM_MSG_VLAN_DELETE_PORT
- NSM_MSG_SVLAN_ADD_CE_PORT
- NSM_MSG_SVLAN_DELETE_CE_PORT
- IF_CALLBACK_VR_BIND
- IF_CALLBACK_VR_UNBIND
- NSM_MSG_VR_SYNC_DONE

EVC/UNI Attributes

- NSM_MSG_EVC_ADD
- NSM_MSG_EVC_DELETE
- NSM_MSG_EVC_ATTIBUTES
- NSM_MSG_EVC_BW_ATTRIBUTES
- NSM_MSG_UNI_ATTRIBUTES
- NSM_MSG_UNI_BW_ATTRIBUTES
- NSM_MSG_CEVLAN_EVC_MAP_INFO

Service Processes

The following describes the service processes for the ELMI protocol.

1. On ELMI startup, ELMI-NSM-client connects with the NSM-server and registers services such as VR, IF information, Bridge information, Port type information, and VLAN information.
2. NSM server, on the client connection, sends the information about the registered services to ELMI
3. On receiving this information from NSM, ELMI updates its own data structures.

EVC/UNI Attributes

EVC/UNI attribute parameters that communicate from the NSM to the ELMI protocol include the following: Communication of UNI and EVC attributes to the CE

UNI attributes:

- UNI Identifier (that is, a user-configured name for UNI)
- CE-VLAN ID/EVC map type (all-to-one bundling, service multiplexing with bundling or no bundling)

Bandwidth profile attributes include the following:

- CM (Coupling Mode)
- CF (Color Flag)

- CIR (Committed Information Rate)
- CBR (Committed Burst Size)
- EIR (Excess Information Rate)
- EBS (Excess Burst Size)

EVC attributes includes the following:

- EVC reference ID
- EVC status type (that is, New, Active, or Not Active)
- EVC type (point-to-point or multipoint-to-multipoint)
- EVC ID (a user-configured name for EVC)

Bandwidth profile attributes that are not supported include:

- CM (coupling mode)
- CF (color flag)
- CIR (committed Information rate)
- CBR (committed burst size)
- EIR (excess information rate)
- EBS (excess burst size
- CE-VLAN ID/EVC map

For the following two message types, the ELMI protocol sends a message to the NSM:

- ELMI_NSM_OPERATIONAL_STATUS
- ELMI_NSM_CEVLAN_ADD_TO_UNIC

Appendix B System Services

The following services are provided by the NSM to the ELMI protocol:

The ELMI-NSM-Client registers with NSM on startup with the following service bits:

- NSM_SERVICE_INTERFACE
- NSM_SERVICE_BRIDGE
- NSM_SERVICE_VLAN
- NSM_SERVICE_EVC
- NSM_MSG_SERVICE_REPLY

The ELMI-NSM-Client module registers for the following messages with NSM:

Link Messages

- NSM_MSG_LINK_UP
- NSM_MSG_LINK_DOWN

Bridge/VLAN Messages

- NSM_MSG_BRIDGE_ADD
- NSM_MSG_BRIDGE_DELETE
- NSM_MSG_BRIDGE_ADD_PORT
- NSM_MSG_BRIDGE_DELETE_PORT
- NSM_MSG_VLAN_PORT_TYPE
- NSM_MSG_VLAN_ADD
- NSM_MSG_VLAN_DELETE
- NSM_MSG_VLAN_ADD_PORT
- NSM_MSG_VLAN_DELETE_PORT
- NSM_MSG_SVLAN_ADD_CE_PORT
- NSM_MSG_SVLAN_DELETE_CE_PORT
- IF_CALLBACK_VR_BIND
- IF_CALLBACK_VR_UNBIND
- NSM_MSG_VR_SYNC_DONE

EVC/UNI Attributes

- NSM_MSG_EVC_ADD
- NSM_MSG_EVC_DELETE
- NSM_MSG_EVC_ATTIBUTES
- NSM_MSG_EVC_BW_ATTRIBUTES
- NSM_MSG_UNI_ATTRIBUTES
- NSM_MSG_UNI_BW_ATTRIBUTES
- NSM_MSG_CEVLAN_EVC_MAP_INFO

Index