Signalling Link Interface (SLI) Application Programming Interface

Version 0.9a Edition 8
Updated 2008-07-03
Distributed with Package strss7-0.9a.8

Abstract

This document is a Application Programming Interface containing technical details concerning the implementation of the Signalling Link Interface (SLI) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the Signalling Link Interface (SLI). It provides abstraction of the signalling link interface to these components as well as providing a basis for signalling link control for other signalling link protocols.

Brian Bidulock <bidulock@openss7.org> for
The OpenSS7 Project <http://www.openss7.org/>

Published by:

OpenSS7 Corporation
1469 Jefferys Crescent
Edmonton, Alberta T6L 6T1
Canada

Notice:

Short Contents

Preface 1
1 Introduction 3
2 The Signalling Link Layer 5
3 SLI Services Definition 9
4 SLI Primitives 27
5 Diagnostics Requirements 129
A LMI Header File Listing 131
B SLI Header File Listing 139
License 149
Glossary 157
Acronyms 159
References 161
Indices 163

Table of Contents

Preface **1**
Security Warning 1
Abstract 1
Purpose 1
Intent 2
Audience 2
Disclaimer 2
Revision History 2

1 Introduction **3**
1.1 Related Documentation 3
1.1.1 Role 3
1.2 Definitions, Acronyms, Abbreviations 3

2 The Signalling Link Layer **5**
2.1 Model of the SLI 5
2.2 SLI Services 6
2.2.1 Local Management 6
2.2.2 Protocol 6
2.3 Purpose of the SLI 7

3 SLI Services Definition **9**
3.1 Local Management Services 9
3.1.1 Acknowledgement Service 9
3.1.2 Information Reporting Service 10
3.1.3 Physical Point of Attachment Service 10
3.1.3.1 PPA Attachment Service 11
3.1.3.2 PPA Detachment Service 11
3.1.4 Initialization Service 12
3.1.4.1 Interface Enable Service 12
3.1.4.2 Interface Disable Service 12
3.1.5 Options Management Service 13
3.1.6 Error Reporting Service 14
3.1.7 Statistics Reporting Service 14
3.1.8 Event Reporting Service 15
3.2 Protocol Services 15
3.2.1 Link Initialization Services 15
3.2.1.1 Power On Service 15
3.2.1.2 Emergency Service 16
3.2.1.3 Start Service 16
3.2.1.4 Stop Service 17
3.2.2 Data Transfer Service 18

3.2.3 Congestion Services ... 18
3.2.3.1 Transmit Congestion Service ... 18
3.2.3.2 Receive Congestion Service ... 19
3.2.4 Restoration Services ... 20
3.2.4.1 BSNT Retrieval Service ... 20
3.2.4.2 Buffer Updating Service ... 21
3.2.4.3 Buffer Clearing Service ... 22
3.2.5 Processor Outage Services ... 23
3.2.5.1 Local Processor Outage Service ... 24
3.2.5.2 Remote Processor Outage Service ... 24
3.2.6 Link Option Management Service ... 25
3.2.7 Event Notification Service ... 26

4 SLI Primitives ... 27

4.1 Local Management Service Primitives ... 27
4.1.1 Acknowledgement Service Primitives ... 27
4.1.1.1 LMI_OK_ACK ... 27
4.1.1.2 LMI_ERROR_ACK ... 29
4.1.2 Information Reporting Service Primitives ... 34
4.1.2.1 LMI_INFO_REQ ... 34
4.1.2.2 LMI_INFO_ACK ... 37
4.1.3 Physical Point of Attachment Service Primitives ... 39
4.1.3.1 LMI_ATTACH_REQ ... 39
4.1.3.2 LMI_DETACH_REQ ... 42
4.1.4 Initialization Service Primitives ... 45
4.1.4.1 LMI_ENABLE_REQ ... 45
4.1.4.2 LMI_ENABLE_CON ... 49
4.1.4.3 LMI_DISABLE_REQ ... 50
4.1.4.4 LMI_DISABLE_CON ... 53
4.1.5 Options Management Service Primitives ... 54
4.1.5.1 LMI_OPTMGMT_REQ ... 54
4.1.5.2 LMI_OPTMGMT_ACK ... 58
4.1.6 Event Reporting Service Primitives ... 60
4.1.6.1 LMI_ERROR_IND ... 60
4.1.6.2 LMI_STATS_IND ... 64
4.1.6.3 LMI_EVENT_IND ... 65
4.2 Protocol Service Primitives ... 66
4.2.1 Link Initialization Service Primitives ... 66
4.2.1.1 SL_POWER_ON_REQ ... 66
4.2.1.2 SL_EMERGENCY_REQ ... 68
4.2.1.3 SL_EMERGENCY_CEASES_REQ ... 70
4.2.1.4 SL_START_REQ ... 72
4.2.1.5 SL_IN_SERVICE_IND ... 75
4.2.1.6 SL_OUT_OF_SERVICE_IND ... 76
4.2.1.7 SL_STOP_REQ ... 78
4.2.2 Data Transfer Service Primitives ... 80
4.2.2.1 SL_PDU_REQ ... 80
4.2.2.2 SL_PDU_IND ... 82

4.2.3 Congestion Service Primitives 83
4.2.3.1 SL_LINK_CONGESTED_IND 83
4.2.3.2 SL_LINK_CONGESTION_CEASED_IND 85
4.2.3.3 SL_CONGESTION_DISCARD_REQ 87
4.2.3.4 SL_CONGESTION_ACCEPT_REQ 89
4.2.3.5 SL_NO_CONGESTION_REQ 91
4.2.4 Restoration Service Primitives 93
4.2.4.1 SL_RETRIEVE_BSNT_REQ 93
4.2.4.2 SL_BSNT_IND 95
4.2.4.3 SL_BSNT_NOT_RETRIEVABLE_IND 96
4.2.4.4 SL_RETRIEVAL_REQUEST_AND_FSNC_REQ 97
4.2.4.5 SL_RETRIEVED_MESSAGE_IND 99
4.2.4.6 SL_RETRIEVAL_COMPLETE_IND 101
4.2.4.7 SL_RETRIEVAL_NOT_POSSIBLE_IND 103
4.2.4.8 SL_CLEAR_BUFFERS_REQ 104
4.2.4.9 SL_CLEAR_RTB_REQ 106
4.2.4.10 SL_RB_CLEARED_IND 108
4.2.4.11 SL_RTB_CLEARED_IND 109
4.2.5 Processor Outage Service Primitives 110
4.2.5.1 SL_LOCAL_PROCESSOR_OUTAGE_REQ 110
4.2.5.2 SL_LOCAL_PROCESSOR_OUTAGE_IND 112
4.2.5.3 SL_RESUME_REQ 113
4.2.5.4 SL_LOCAL_PROCESSOR_RECOVERED_IND 115
4.2.5.5 SL_REMOTE_PROCESSOR_OUTAGE_IND 116
4.2.5.6 SL_REMOTE_PROCESSOR_RECOVERED_IND 117
4.2.5.7 SL_CONTINUE_REQ 118
4.2.6 Link Option Management Service Primitives 120
4.2.6.1 SL_OPTMGMT_REQ 120
4.2.6.2 SL_OPTMGMT_ACK 124
4.2.7 Event Notification Service Primitives 126
4.2.7.1 SL_NOTIFY_REQ 126
4.2.7.2 SL_NOTIFY_IND 128

5 Diagnostics Requirements 129

5.1 Non-Fatal Error Handling Facility 129
5.2 Fatal Error Handling Facility 129

Appendix A LMI Header File Listing 131

Appendix B SLI Header File Listing 139

License 149

GNU Free Documentation License 149
Preamble 149
Terms and Conditions for Copying, Distribution and Modification 149
How to use this License for your documents 155

Glossary **157**

Acronyms **159**

References **161**

Indices **163**

Concept Index 163
Type Index 164
Variable Index 165
Primitive Index 166
Primitive Value Index 167
Protocol State Index 168
Protocol Error Index 169
Manual Page Index 170

Preface

Security Warning

Abstract

This document is a Application Programming Interface containing technical details concerning the implementation of the Signalling Link Interface (SLI) for OpenSS7. It contains recommendations on software architecture as well as platform and system applicability of the Signalling Link Interface (SLI).

This document specifies a Signalling Link Interface (SLI) Specification in support of the OpenSS7 Signalling Link (SL) protocol stacks. It provides abstraction of the signalling link interface to these components as well as providing a basis for signalling link control for other link control protocols.

Purpose

The purpose of this document is to provide technical documentation of the Signalling Link Interface (SLI). This document is intended to be included with the OpenSS7 *STREAMS* software package released by *OpenSS7 Corporation*. It is intended to assist software developers, maintainers and users of the Signalling Link Interface (SLI) with understanding the software architecture and technical interfaces that are made available in the software package.

Intent

It is the intent of this document that it act as the primary source of information concerning the Signalling Link Interface (SLI). This document is intended to provide information for writers of OpenSS7 Signalling Link Interface (SLI) applications as well as writers of OpenSS7 Signalling Link Interface (SLI) Users.

Audience

The audience for this document is software developers, maintainers and users and integrators of the Signalling Link Interface (SLI). The target audience is developers and users of the OpenSS7 SS7 stack.

Disclaimer

Although the author has attempted to ensure that the information in this document is complete and correct, neither the Author nor OpenSS7 Corporation will take any responsibility in it.

Revision History

Take care that you are working with a current version of this documentation: you will not be notified of updates. To ensure that you are working with a current version, check the OpenSS7 Project website for a current version.

Only the texinfo or roff source is controlled. A printed (or postscript) version of this document is an **UNCONTROLLED VERSION**.

```
$Log: sli.texi,v $
Revision 0.9.2.5  2008-04-29 07:10:40  brian
- updating headers for release

Revision 0.9.2.4  2007/08/14 12:17:05  brian
- GPLv3 header updates

Revision 0.9.2.3  2007/08/03 13:34:54  brian
- manual updates, put ss7 modules in public release

Revision 0.9.2.2  2007/07/09 09:04:51  brian
- working up SLI specification

Revision 0.9.2.1  2007/07/04 08:24:58  brian
- added new files
```

1 Introduction

This document specifies a *STREAMS*-based kernel-level instantiation of the ITU-T Signalling Link Interface (SLI) definition. The Signalling Link Interface (SLI) enables the user of a a signalling link service to access and use any of a variety of conforming signalling link providers without specific knowledge of the provider's protocol. The service interface is designed to support any network signalling link protocol and user signalling link protocol. This interface only specifies access to signalling link service providers, and does not address issues concerning signalling link management, protocol performance, and performance analysis tools.

This specification assumes that the reader is familiar with ITU-T state machines and signalling link interfaces (e.g. Q.703, Q.2210), and *STREAMS*.

1.1 Related Documentation

— **ITU-T Recommendation Q.703 (White Book)**
— **ITU-T Recommendation Q.2210 (White Book)**
— **ANSI T1.111.3/2002**
— **System V Interface Definition, Issue 2 - Volume 3**

1.1.1 Role

This document specifies an interface that supports the services provided by the *Signalling System No. 7 (SS7)* for ITU-T, ANSI and ETSI applications as described in ITU-T Recommendation Q.703, ITU-T Recommendation Q.2210, ANSI T1.111.3, ETSI ETS 300 008-1. These specifications are targeted for use by developers and testers of protocol modules that require signalling link service.

1.2 Definitions, Acronyms, Abbreviations

LM Local Management.

LMS Local Management Service.

LMS User A user of Local Management Services.

LMS Provider
A provider of Local Management Services.

Originating SL User
A SL-User that initiates a Signalling Link.

Destination SL User
A SL-User with whom an originating SL user wishes to establish a Signalling Link.

ISO International Organization for Standardization

SL User Kernel level protocol or user level application that is accessing the services of the Signalling Link sub-layer.

SL Provider
: Signalling Link sub-layer entity/entities that provide/s the services of the Signalling Link interface.

SLI
: Signalling Link Interface

TIDU
: Signalling Link Interface Data Unit

TSDU
: Signalling Link Service Data Unit

OSI
: Open Systems Interconnection

QOS
: Quality of Service

STREAMS
: A communication services development facility first available with UNIX System V Release 3.

2 The Signalling Link Layer

The Signalling Link Layer provides the means to manage the association of SL-Users into connections. It is responsible for the routing and management of data to and from signalling link connections between SL-user entities.

2.1 Model of the SLI

The SLI defines the services provided by the signalling link layer to the signalling link user at the boundary between the signalling link provider and the signalling link user entity. The interface consists of a set of primitives defined as *STREAMS* messages that provide access to the signalling link layer services, and are transferred between the SLS user entity and the SLS provider. These primitives are of two types; ones that originate from the SLS user, and other that originate from the SLS provider. The primitives that originate from the SLS user make requests to the SLS provider, or respond to an indication of an event of the SLS provider. The primitives that originate from the SLS provider are either confirmations of a request or are indications to the CCS user that an event has occurred. Figure 2.1 shows the model of the SLI.

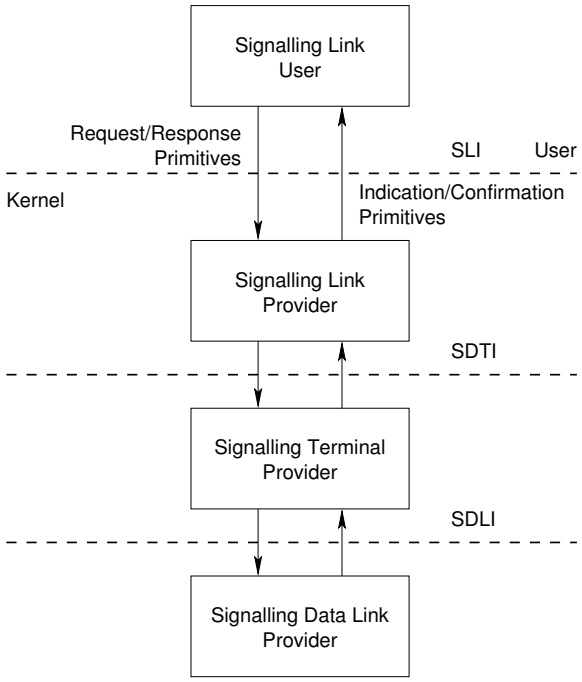

Figure 2.1: *Model of the SLI*

The SLI allows the SLS provider to be configured with any signalling link layer user (such as a signalling link application) that also conforms to the SLI. A signalling link layer user can also be a user program that conforms to the SLI and accesses the SLS provider via

putmsg(2s) and **getmsg**(2s) system calls. The typical configuration, however, is to link a signalling link stream beneath a message transfer part multiplexing driver.

2.2 SLI Services

The features of the SLI are defined in terms of the services provided by the SLS provider, and the individual primitives that may flow between the SLS user and the SLS provider.

The SDLI Services are broken into two groups: local management services and protocol services. Local management services are responsible for the local management of streams, assignment of streams to physical points of attachment, enabling and disabling of streams, management of options associated with a stream, and general acknowledgement and event reporting for the stream. Protocol services consist of .

2.2.1 Local Management

Local management services are listed in Table 2.1.

Phase	Service	Primitives
Local Management	Acknowledgement	LMI_OK_ACK, LMI_ERROR_ACK
	Information Reporting	LMI_INFO_REQ, LMI_INFO_ACK
	PPA Attachment	LMI_ATTACH_REQ, LMI_DETACH_REQ, LMI_OK_ACK
	Initialization	LMI_ENABLE_REQ, LMI_ENABLE_CON, LMI_DISABLE_REQ, LMI_DISABLE_CON
	Options Management	LMI_OPTMGMT_REQ, LMI_OPTMGMT_ACK
	Event Reporting	LMI_ERROR_IND, LMI_STATS_IND, LMI_EVENT_IND

Table 2.1: *Local Management Services*

The local management services interface is described in Section 3.1 [Local Management Services], page 9, and the primitives are detailed in Section 4.1 [Local Management Service Primitives], page 27. The local management services interface is defined by the '`ss7/lmi.h`' header file (see Appendix A [LMI Header File Listing], page 131).

2.2.2 Protocol

Protocol services are listed in Table 2.2.

Phase	Service	Primitives
Initialization	Power On	SL_POWER_ON_REQ,
	Emergency	SL_EMERGENCY_REQ, SL_EMERGENCY_CEASES_REQ,
	Start	SL_START_REQ, SL_IN_SERVICE_IND,
	Stop	SL_OUT_OF_SERVICE_IND, SL_STOP_REQ,
Data Transfer	Data Transfer	SL_PDU_REQ, SL_PDU_IND
Congestion	Transmit Congestion	SL_LINK_CONGESTED_IND, SL_LINK_CONGESTION_CEASED_IND
	Receive Congestion	SL_CONGESTION_DISCARD_REQ, SL_CONGESTION_ACCEPT_REQ, SL_NO_CONGESTION_REQ
Restoration	BSNT Retrieval	SL_RETRIEVE_BSNT_REQ, SL_BSNT_IND, SL_BSNT_NOT_RETRIEVABLE_IND
	Buffer Updating	SL_RETRIEVAL_REQUEST_AND_FSNC_REQ, SL_RETRIEVED_MESSAGE_IND, SL_RETRIEVAL_COMPLETE_IND, SL_RETRIEVAL_NOT_POSSIBLE_IND
	Buffer Clearing	SL_CLEAR_BUFFERS_REQ, SL_CLEAR_RTB_REQ, SL_RB_CLEARED_IND, SL_RTB_CLEARED_IND
Processor Outage	Local Processor Outage	SL_LOCAL_PROCESSOR_OUTAGE_REQ, SL_LOCAL_PROCESSOR_OUTAGE_IND, SL_RESUME_REQ, SL_LOCAL_PROCESSOR_RECOVERED_IND
	Remote Processor Outage	SL_REMOTE_PROCESSOR_OUTAGE_IND, SL_REMOTE_PROCESSOR_RECOVERED_IND, SL_CONTINUE_REQ
Options Management	Options Management	SL_OPTMGMT_REQ, SL_OPTMGMT_ACK
Event Notification	Event Notification	SL_NOTIFY_REQ, SL_NOTIFY_IND

Table 2.2: *Protocol Services*

The protocol services interface is described in Section 3.2 [Protocol Services], page 15, and the primitives are detailed in Section 4.2 [Protocol Service Primitives], page 66. The protocol services interface is defined by the '`ss7/sli.h`' header file (see Appendix B [SLI Header File Listing], page 139).

2.3 Purpose of the SLI

The SLI is typically implemented as a device driver controlling an intelligent protocol controller device that provides access to channels. The purpose behind exposing this low level interface is that almost all communications channel devices can be placed into a SS7 HDLC mode, where a data stream can be exchanged between the driver and the medium. The SLI

provides and inteface that, once implemented as a driver for a new device, can provide complete and verified SS7 signalling link capabilities by linking under a generic MTP (Message Transfer Part) multiplex driver an open device stream.

This allows MTP drivers to be verified independently for correct operation and then simply used for all manner of new device drivers that can implement the SLI interface.

3 SLI Services Definition

3.1 Local Management Services

3.1.1 Acknowledgement Service

The acknowledgement service provides the LMS user with the ability to receive positive and negative acknowledgements regarding the successful or unsuccessful completion of services.

- `LMI_OK_ACK`: The `LMI_OK_ACK` message is used by the LMS provider to indicate successful receipt and completion of a service primitive request that requires positive acknowledgement.
- `LMI_ERROR_ACK`: The `LMI_ERROR_ACK` message is used by the LMS provider to indicate successful receipt and failure to complete a service primitive request that requires negative acknowledgement.

A successful invocation of the acknowledgement service is illustrated in Figure 3.1.

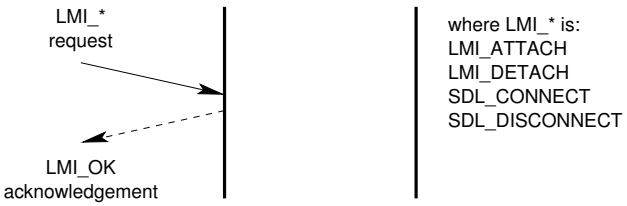

Figure 3.1: *Message Flow: Successful Acknowledgement Service*

As illustrated in Figure 3.1, the service primitives for which a positive acknowledgement may be returned are the `LMI_ATTACH_REQ` and `LMI_DETACH_REQ`.

An unsuccessful invocation of the acknowledgement service is illustrated in Figure 3.2.

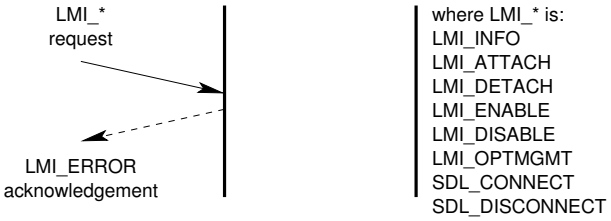

Figure 3.2: *Message Flow: Unsuccessful Acknowledgement Service*

As illustrated in Figure 3.2, the service primitives for which a negative acknowledgement may be returned are the `LMI_INFO_REQ`, `LMI_ATTACH_REQ`, `LMI_DETACH_REQ`, `LMI_ENABLE_REQ`, `LMI_DISABLE_REQ` and `LMI_OPTMGMT_REQ` messages.

3.1.2 Information Reporting Service

The information reporting service provides the LMS user with the ability to elicit information from the LMS provider.

- `LMI_INFO_REQ`: The `LMI_INFO_REQ` message is used by the LMS user to request information about the LMS provider.
- `LMI_INFO_ACK`: The `LMI_INFO_ACK` message is issued by the LMS provider to provide requested information about the LMS provider.

A successful invocation of the information reporting service is illustrated in Figure 3.3.

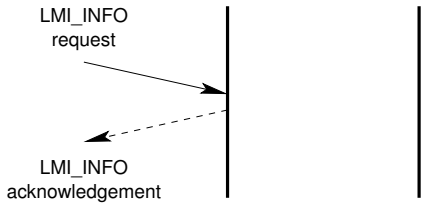

Figure 3.3: *Message Flow: Successful Information Reporting Service*

3.1.3 Physical Point of Attachment Service

The local management interface provides the LMS user with the ability to associate a stream to a physical point of appearance (*PPA*) or to disassociate a stream from a PPA. The local management interface provides for two styles of LMS provider:

Style 1 LMS Provider

A *Style 1* LMS provider is a provider that associates a stream with a PPA at the time of the first `open(2)` call for the device, and disassociates a stream from a PPA at the time of the last `close(2)` call for the device.

Physical points of attachment (PPA) are assigned to major and minor device number combinations. When the major and minor device number combination is opened, the opened stream is automatically associated with the PPA for the major and minor device number combination. The last close of the device disassociates the PPA from the stream.

Freshly opened *Style 1* LMS provider streams start life in the `LMI_DISABLED` state.

This approach is suitable for LMS providers implemented as real or pseudo-device drivers and is applicable when the number of minor devices is small and static.

Style 2 LMS Provider

A *Style 2* LMS provider is a provider that associates a stream with a PPA at the time that the LMS user issues the `LMI_ATTACH_REQ` message. Freshly opened streams are not associated with any PPA. The *Style 2* LMS provider stream is disassociated from a PPA when the stream is closed or when the LMS user issues the `LMI_DETACH_REQ` message.

Freshly opened *Style 2* LMS provider streams start life in the `LMI_UNATTACHED` state.

This approach is suitable for LMS providers implemented as clone real or pseudo-device drivers and is applicable when the number of minor devices is large or dynamic.

3.1.3.1 PPA Attachment Service

The PPA attachment service provides the LMS user with the ability to attach a *Style 2* LMS provider stream to a physical point of appearance (PPA).

- `LMI_ATTACH_REQ`: The `LMI_ATTACH_REQ` message is issued by the LMS user to request that a *Style 2* LMS provider stream be attached to a specified physical point of appearance (PPA).
- `LMI_OK_ACK`: Upon successful receipt and processing of the `LMI_ATTACH_REQ` message, the LMS provider acknowledges the success of the service completion with a `LMI_OK_ACK` message.
- `LMI_ERROR_ACK`: Upon successful receipt but failure to process the `LMI_ATTACH_REQ` message, the LMS provider acknowledges the failure of the service completion with a `LMI_ERROR_ACK` message.

A successful invocation of the attachment service is illustrated in Figure 3.4.

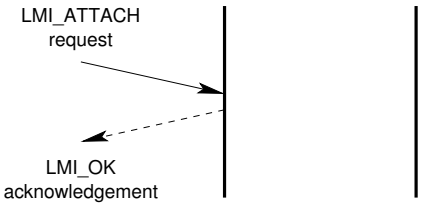

Figure 3.4: *Message Flow: Successful Attachment Service*

3.1.3.2 PPA Detachment Service

The PPA detachment service provides the LMS user with the ability to detach a *Style 2* LMS provider stream from a physical point of attachment (PPA).

- `LMI_DETACH_REQ`: The `LMI_DETACH_REQ` message is issued by the LMS user to request that a *Style 2* LMS provider stream be detached from the attached physical point of appearance (PPA).
- `LMI_OK_ACK`: Upon successful receipt and processing of the `LMI_DETACH_REQ` message, the LMS provider acknowledges the success of the service completion with a `LMI_OK_ACK` message.
- `LMI_ERROR_ACK`: Upon successful receipt but failure to process the `LMI_DETACH_REQ` message, the LMS provider acknowledges the failure of the service completion with a `LMI_ERROR_ACK` message.

A successful invocation of the detachment service is illustrated in Figure 3.5.

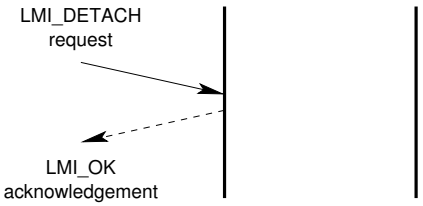

Figure 3.5: *Message Flow: Successful Detachment Service*

3.1.4 Initialization Service

The initialization service provides the LMS user with the abilty to enable and disable the stream for the associated PPA.

3.1.4.1 Interface Enable Service

The interface enable service provides the LMS user with the ability to enable an LMS provider stream that is associated with a PPA. Enabling the interface permits the LMS user to exchange protocol service interface messages with the LMS provider.

- `LMI_ENABLE_REQ`: The `LMI_ENABLE_REQ` message is issued by the LMS user to request that the protocol service interface be enabled.
- `LMI_ENABLE_CON`: Upon successful enabling of the protocol service interface, the LMS provider acknowledges successful completion of the service by issuing a `LMI_ENABLE_CON` message to the LMS user.
- `LMI_ERRORK_ACK`: Upon unsuccessful enabling of the protocol service interface, the LMS provider acknowledges the failure to complete the service by issuing an `LMI_ERROR_ACK` message to the LMS user.

A successful invocation of the enable service is illustrated in Figure 3.6.

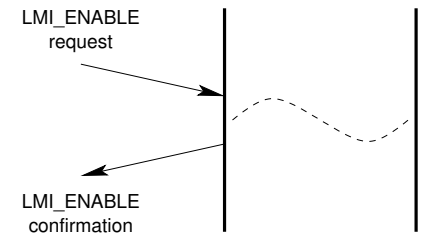

Figure 3.6: *Message Flow: Successful Enable Service*

3.1.4.2 Interface Disable Service

The interface disable service provides the LMS user with the ability to disable an LMS provider stream that is associated with a PPA. Disabling the interface withdraws the LMS user's ability to exchange protocol service interface messages with the LMS provider.

- `LMI_DISABLE_REQ`: The `LMI_DISABLE_REQ` message is issued by the LMS user to request that the protocol service interface be disabled.
- `LMI_DISABLE_CON`: Upon successful disabling of the protocol service interface, the LMS provider acknowledges successful completion of the service by issuing a `LMI_DISABLE_CON` message to the LMS user.
- `LMI_ERRORK_ACK`: Upon unsuccessful disabling of the protocol service interface, the LMS provider acknowledges the failure to complete the service by issuing an `LMI_ERROR_ACK` message to the LMS user.

A successful invocation of the disable service is illustrated in Figure 3.7.

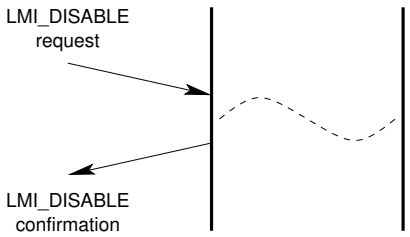

Figure 3.7: *Message Flow: Successful Disable Service*

3.1.5 Options Management Service

The options management service provides the LMS user with the ability to control and affect various generic and provider-specific options associated with the LMS provider.

- `LMI_OPTMGMT_REQ`: The LMS user issues a `LMI_OPTMGMT_REQ` message when it wishes to interrogate or affect the setting of various generic or provider-specific options associated with the LMS provider for the stream upon which the message is issued.
- `LMI_OPTMGMT_ACK`: Upon successful receipt of the `LMI_OPTMGMT_REQ` message, and successful options processing, the LMS provider acknowledges the successful completion of the service with an `LMI_OPTMGMT_ACK` message.
- `LMI_ERROR_ACK`: Upon successful receipt of the `LMI_OPTMGMT_REQ` message, and unsuccessful options processing, the LMS provider acknowledges the failure to complete the service by issuing an `LMI_ERROR_ACK` message to the LMS user.

A successful invocation of the options management service is illustrated in Figure 3.8.

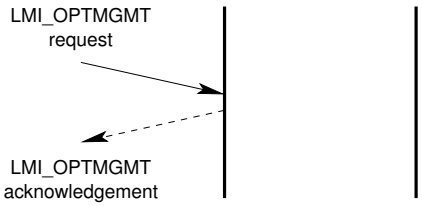

Figure 3.8: *Message Flow: Successful Options Management Service*

3.1.6 Error Reporting Service

The error reporting service provides the LMS provider with the ability to indicate asynchronous errors to the LMS user.

- `LMI_ERROR_IND`: The LMS provider issues the `LMI_ERROR_IND` message to the LMS user when it needs to indicate an asynchronous error (such as the unusability of the communications medium).

A successful invocation of the error reporting service is illustrated in Figure 3.9.

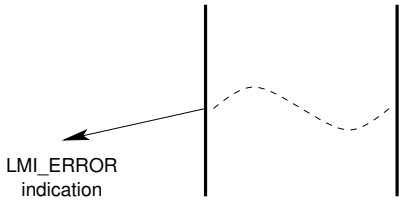

Figure 3.9: *Message Flow: Successful Error Reporting Service*

3.1.7 Statistics Reporting Service

- `LMI_STATS_IND`:

A successful invocation of the statistics reporting service is illustrated in Figure 3.10.

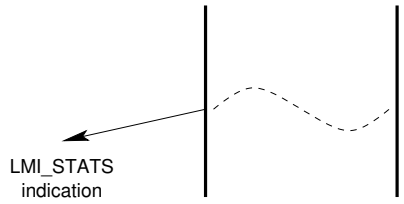

Figure 3.10: *Message Flow: Successful Statistics Reporting Service*

3.1.8 Event Reporting Service

The event reporting service provides the LMS provider with the ability to indicate specific asynchronous management events to the LMS user.

- `LMI_EVENT_IND`: The LMS provider issues the `LMI_EVENT_IND` message to the LMS user when it wishes to indicate an asynchronous (management) event to the LMS user.

A successful invocation of the event reporting service is illustrated in Figure 3.11.

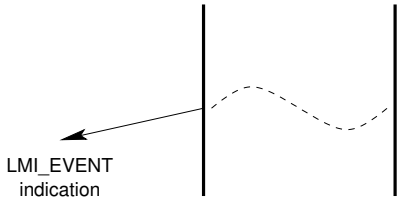

Figure 3.11: *Message Flow: Successful Event Reporting Service*

3.2 Protocol Services

Protocol services are specific to the Signalling Link interface. These services consist of initialization of the link and preparation for the transfer of signal units, the transfer of signal units, transmit and receive congestion control, BSNT retrieval, buffer updating, buffer clearing, local processor outage, remote processor outage, link options management and management event notification.

The service primitives that implement the protocol services are described in detail in Section 4.2 [Protocol Service Primitives], page 66.

3.2.1 Link Initialization Services

The link initialization services provide the SLS user with the ability to power on the terminal, set emergency status, start the signalling link and stop the signalling link. The service primitives that implement the link initialization services are described in detail in Section 4.2.1 [Link Initialization Service Primitives], page 66.

3.2.1.1 Power On Service

The power on service provides the SLS user with the ability to power on the signalling data terminal. The signalling data terminal must be powered on at least once before the link can be started.

- `SL_POWER_ON_REQ`: The `SL_POWER_ON_REQ` message is used by the SLS user to request that the SLS provider power on the signalling data terminal. If the signalling data terminal does not require power (such as a software module), this serves to initialize the signalling data terminal functions.

A successful invocation of the power on service is illustrated in Figure 3.12.

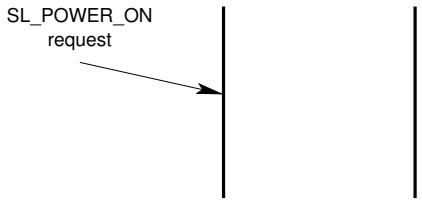

Figure 3.12: *Message Flow: Successful Power On Service*

3.2.1.2 Emergency Service

The emergency service provides the SLS user with the ability to specify whether normal or emergency alignment procedures should take effect on the current or next alignment procedure. Emergency alignment procedures have a shorter duration (short proving period) than normal alignment procedures. Some SS7 protocol variants (TTC) always use emergency alignment procedures and are not affected by this service.

- `SL_EMERGENCY_REQ`: The `SL_EMERGENCY_REQ` message is used by the SLS user to request that the emergency alignment procedure should take effect on the current or next alignment of the signalling link.
- `SL_EMERGENCY_CEASES_REQ`: The `SL_EMERGENCY_CEASES_REQ` message is used by the SLS user to request that the normal alignment procedure should take effect on the current or next alignment of the signalling link.

A successful invocation of the emergency service is illustrated in Figure 3.13.

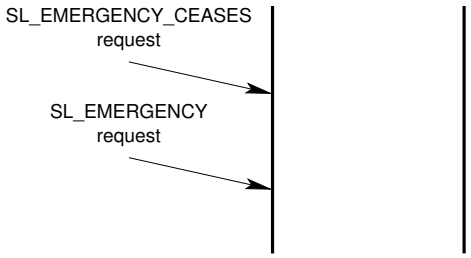

Figure 3.13: *Message Flow: Successful Emergency Service*

3.2.1.3 Start Service

The start service provides the SLS user with the ability to align the signalling link and have it placed into service. The start service must be successfully invoked on both sides of the signalling link before the signalling link is able to exchange message signal units.

- `SL_START_REQ`: The `SL_START_REQ` message is used by the SLS user to request that the signalling link be aligned a placed into service.
- `SL_IN_SERVICE_IND`: The `SL_IN_SERVICE_IND` message is used by the SLS provider

to indicate that the signalling link has been successfully aligned and has been placed into service at Level 2.

A successful invocation of the start service is illustrated in Figure 3.14.

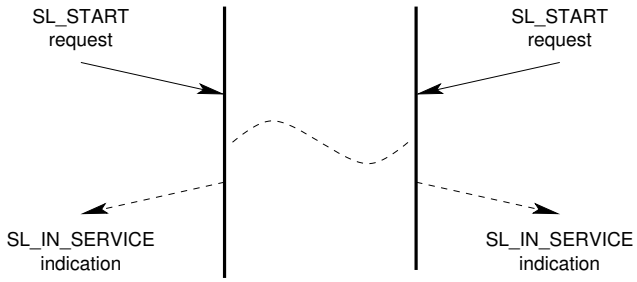

Figure 3.14: *Message Flow: Successful Start Service*

A unsuccessful invocation of the start service is illustrated in Figure 3.15.

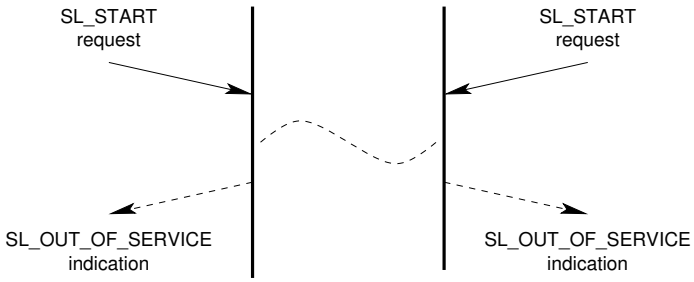

Figure 3.15: *Message Flow: Unsuccessful Start Service*

3.2.1.4 Stop Service

The stop service provides the SLS user and provider with the ability to take a signalling link out of service. Once the stop service has successfully completed, the signalling link is no longer able to exchange message signal units.

- `SL_STOP_REQ`: The `SL_STOP_REQ` message is used by the SLS user to request that the signalling link be taken out of service.
- `SL_OUT_OF_SERVICE_IND`: The `SL_OUT_OF_SERVICE_IND` message is used by the SLS provider to indicate that the signalling link has been taken out of service by the SLS provider.

A successful invocation of the stop service is illustrated in Figure 3.16.

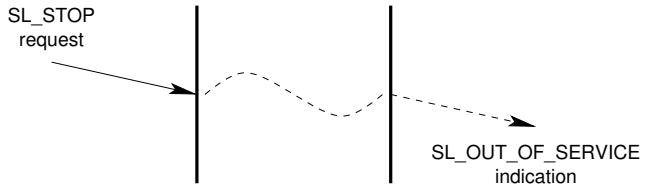

Figure 3.16: *Message Flow: Successful Stop Service*

3.2.2 Data Transfer Service

The data transfer service provides the SLS user with the ability to exchange message signal units on the signalling link. The service primitives that implement the data transfer service are described in detail in Section 4.2.2 [Data Transfer Service Primitives], page 80.

- `SL_PDU_REQ`: The `SL_PDU_REQ` message is used by the SLS user to request that a message signal unit be sent on the signalling link.
- `SL_PDU_IND`: The `SL_PDU_IND` message is used by the SLS provider to indicate that a message signal unit has been received on the signalling link.

A successful invocation of the data transfer service is illustrated in Figure 3.17.

Figure 3.17: *Message Flow: Successful Data Transfer Service*

3.2.3 Congestion Services

The congestion services provide the SLS user with the ability to invoke a receive congestion policy. They also provide the SLS provider with the ability to indicate transmit congestion levels. The service primitives that implement the congestion services are described in detail in Section 4.2.3 [Congestion Service Primitives], page 83.

3.2.3.1 Transmit Congestion Service

The transmit congestion service provides the SLS provider with the ability to indicate transmit congestion (and corresponding levels) to the SLS user. There are 4 levels of congestion, 0, 1, 2 and 3. Each congestion level has an onset threshold and an abatement threshold. When the transmit buffer occupancy exceeds the onset threshold for the level, congestion is indicated at that level. When the transmit buffer occupancy falls below the abatement threshold for the level, congestion abatement is indicated. Some SS7 protocol variants do not have congestion levels and only signal the presence or lack of congestion.

When congestion is indicated at a level, the SLS user should discard messages that have a message priority that is less than the level at which congestion has been indicated.

- `SL_LINK_CONGESTED_IND`: The `SL_LINK_CONGESTED_IND` message is used by the SLS provider to indicate that congestion onset has occurred for the congestion level indicated in the message.
- `SL_LINK_CONGESTION_CEASED_IND`: The `SL_LINK_CONGESTION_CEASED_IND` message is used by the SLS provider to indicate that congestion abatement has occurred for the congestion level indicated in the message.

A successful indication of the transmit congestion service is illustrated in Figure 3.18.

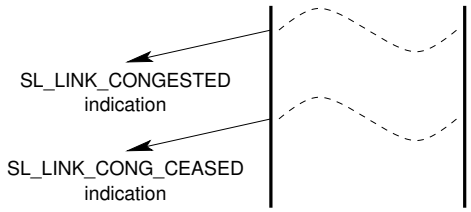

Figure 3.18: *Message Flow: Successful Transmit Congestion Service*

3.2.3.2 Receive Congestion Service

The receive congestion service provides the SLS user with the ability to specify that receive congestion is in effect or has abated and the policy to use for received message signal units under congestion. A discard policy indicates that received message signal units should be discarded (and not acknowledged); receive congestion is signalled to the sending side of the signalling link. An accept policy indicates that received message signal units should not be discarded and should be acknowledged; receive congestion is signalled to the sending side of the signalling link. When receive congestion abates, the abatement of receive congestion is signalled to the sending side of the signalling link.

The SLS provider may also perform its own receive congestion onset, abatement and policy. The SLS provider does not indicate its current receive congestion level or policy to the SLS user.

- `SL_NO_CONGESTION_REQ`: The `SL_NO_CONGESTION_REQ` message is used by the SLS user to specify that receive congestion has abated and that receive congestion should no longer be signalled to the sending side of the signalling link.
- `SL_CONGESTION_ACCEPT_REQ`: The `SL_CONGESTION_ACCEPT_REQ` message is used by the SLS user to specify that receive congestion has onset and that receive congestion should be signalled to the sending side of the signalling link. The congestion policy is an accept policy that allows message signal units to continue to be delivered to the SLS user and acknowledged to the remote end of the signalling link.
- `SL_CONGESTION_DISCARD_REQ`: The `SL_CONGESTION_DISCARD_REQ` message is used by the SLS user to specify that receive congestion has onset and that receive congestion

should be signalled to the sending side of the signalling link. The congestion policy is a discard policy that requires the SLS provider to discard message signal units without delivering them to the SLS user and they are not to be acknowledged to the remote end of the signalling link.

A successful invocation of the receive congestion service is illustrated in Figure 3.19.

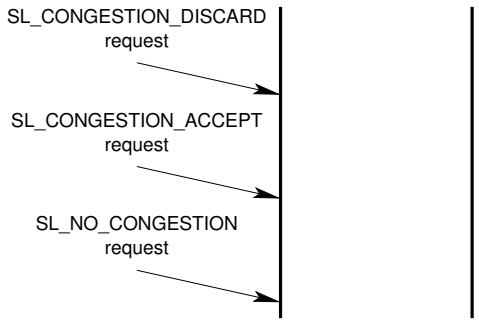

Figure 3.19: *Message Flow: Successful Receive Congestion Service*

3.2.4 Restoration Services

Restoration services consist of the services necessary to change over a link, update its buffers, and clearing any unnecessarily old MSUs from the receive buffer or retransmission buffer. The service primitives that implement the restoration services are detailed in Section 4.2.4 [Restoration Service Primitives], page 93.

3.2.4.1 BSNT Retrieval Service

The BSNT retrieval service is a somewhat optional service in support of the sequenced changeover procedure of the Message Transfer Part. It is '`somewhat`' optional due to the possibility that time-controlled changeover is always used, per ETSI ETS 300 008-1.

- `SL_RETRIEVE_BSNT_REQ`: The `SL_RETRIEVE_BSNT_REQ` message is used by the SLS user to request that the SLS provider indicate the last transmitted backward sequence number (BSNT).
- `SL_BSNT_IND`: The `SL_BSNT_IND` message is used by the SLS provider to indicate the last transmitted backward sequence number (BSNT) when requested by the SLS user with a `SL_RETRIEVE_BSNT_REQ` message.
- `SL_BSNT_NOT_RETRIEVABLE_IND`: The `SL_BSNT_NOT_RETRIEVABLE_IND` message is used by the SLS provider to indicate that the last transmitted backward sequence number (BSNT) is not available when requested by the SLS user with a `SL_RETRIEVE_BSNT_REQ` message. This may be due to hardware or other failures.

A successful invocation of the BSNT retrieval service is illustrated in Figure 3.20.

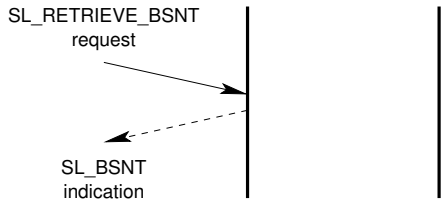

Figure 3.20: *Message Flow: Successful BSNT Retreival Service*

An unsuccessful invocation of the BSNT retrieval service is illustrated in Figure 3.21.

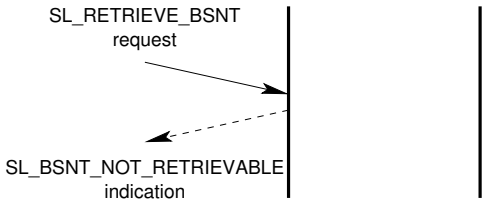

Figure 3.21: *Message Flow: Unsuccessful BSNT Retrieval Service*

3.2.4.2 Buffer Updating Service

The buffer updating service provides the SLS user with the ability to update the retransmission buffer and collect messages that have not been successfully received by the remote side of the signalling link during a sequenced changeover procedure. The SLS user specifies the FSNC (the forward sequence number confirmed received by the remote end of the signalling link). The SLS provider uses the FSNC to purge successfully received messages from the retransmission buffer and then indicates the remaining contents of the retransmission buffer and the transmission buffer to the SLS user.

The SLS user may also clear the retransmission buffer using the buffer clearing service before retrieving messages. This this case, the messages retrieved by the SLS provider will be the contents of the transmission buffer. The combination of the two services are used to perform the time controlled changeover procedure.

- `SL_RETRIEVAL_REQUEST_AND_FSNC_REQ`: The `SL_RETRIEVAL_REQUEST_AND_FSNC_REQ` message is used by the SLS user to request the SLS provider update the retransmission buffer to reflect the value of the specified FSNC and retrieve and indicate the contents of the updated retransmission buffer followed by the contents of the transmission buffer to the SLS user.
- `SL_RETRIEVED_MESSAGE_IND`: The `SL_RETRIEVED_MESSAGE_IND` message is used by the SLS provider to indicate one message from the retransmission buffer or transmission buffer.
- `SL_RETRIEVAL_COMPLETE_IND`: The `SL_RETRIEVAL_COMPLETE_IND` message is used by the SLS provider to indicate that the retrieval of messages from the retransmission buffer and transmission buffer is complete.

- `SL_RETRIEVAL_NOT_POSSIBLE_IND`: The `SL_RETRIEVAL_NOT_POSSIBLE_IND` message is used by the SLS provider to indicate that the updating of the retransmission buffer to the specified FSNC and retrieval of messages from the retransmission buffer and transmission buffer is not possible. This may be due to hardware failure.

A successful invocation of the buffer updating service is illustrated in Figure 3.22.

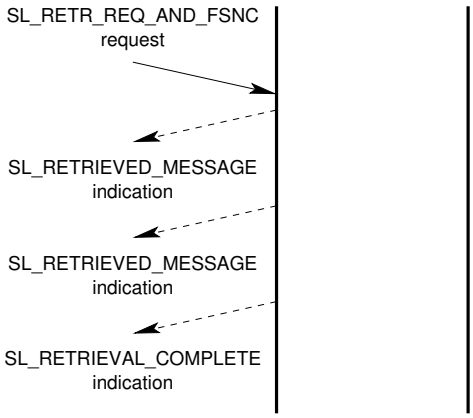

Figure 3.22: *Message Flow: Successful Buffer Updating Service*

An unsuccessful invocation of the buffer updating service is illustrated in Figure 3.23.

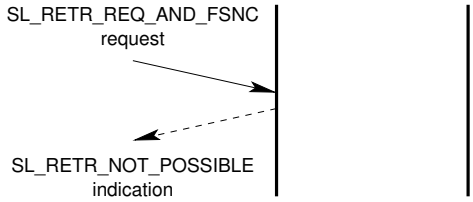

Figure 3.23: *Message Flow: Unsuccessful Buffer Updating Service*

3.2.4.3 Buffer Clearing Service

The buffer clearing service provides the SLS user with the ability to request that all message buffers be cleared (receive buffer, retransmission buffer, transmission buffer) and that the SLS provider indicate when the receive and retransmission buffer are cleared. It also provides the SLS user with the ability to clear only the retransmission buffer and receive and indication when the buffer is cleared.

Clearing of all buffers is performed when the signalling link has been blocked (local or remote processor outage) for a long duration and messages contained in the buffer are too old to be processed.

Clearing of the retransmission buffer is performed as part of the time-controlled changeover procedure, when the value of the FSNC has not been received in a sequenced changeover message from the adjacent signalling point.

- `SL_CLEAR_BUFFERS_REQ`: The `SL_CLEAR_BUFFERS_REQ` message is used by the SLS user to request that all message buffers (receive, retransmit, transmit) be cleared.
- `SL_CLEAR_RTB_REQ`: The `SL_CLEAR_RTB_REQ` message is used by the SLS user to request that only the retransmission buffer be cleared as part of a time-controlled changeover procedure.
- `SL_RB_CLEARED_IND`: The `SL_RB_CLEARED_IND` message is used by the SLS provider to indicate when the receive buffer has been successfully cleared.
- `SL_RTB_CLEARED_IND`: The `SL_RTB_CLEARED_IND` message is used by the SLS provider to indicate when the retransmission buffer has been successfully cleared.

A successful invocation of the buffer clearing service is illustrated in Figure 3.24 and Figure 3.25.

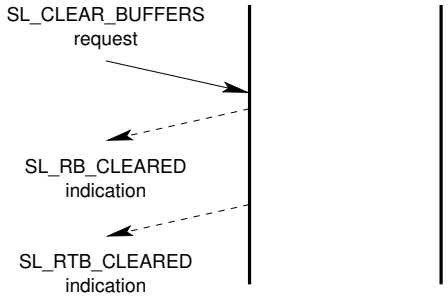

Figure 3.24: *Message Flow: Successful Buffer Clearing Service*

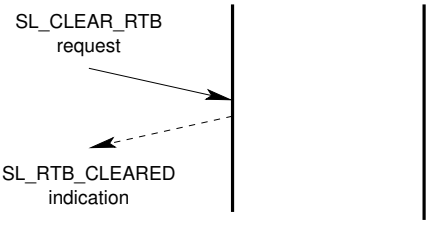

Figure 3.25: *Message Flow: Successful Buffer Clearing Service*

3.2.5 Processor Outage Services

The processor outage services provide the SLS user with the ability to request a local processor outage as well as being informed of a local or remote processor outage. The service primitives that implement the processor outage services are described in detail in Section 4.2.5 [Processor Outage Service Primitives], page 110.

3.2.5.1 Local Processor Outage Service

The local processor outage service provides the SLS user with the ability to both request a local processor outage as well as be informed of a local processor outage. Local processor outage occurs when the SLS user is unable to pass message signal units for transmission or accept received message signal units, or the SLS provider is unable to deliver received message signal units or accept message signal units for transmission. Local processor outage conditions can exist independently within the SLS user and within the SLS provider.

- `SL_LOCAL_PROCESSOR_OUTAGE_REQ`: The `SL_LOCAL_PROCESSOR_OUTAGE_REQ` message is used by the SLS user to specify that a local processor outage condition exists due to a condition within the SLS user.
- `SL_LOCAL_PROCESSOR_OUTAGE_IND`: The `SL_LOCAL_PROCESSOR_OUTAGE_IND` message is used by the SLS provider to indicate that a local processor outage condition exists due to a condition within the SLS provider.
- `SL_RESUME_REQ`: The `SL_RESUME_REQ` message is used by the SLS use to specify that a local processor outage condition no longer exists within the SLS user.
- `SL_LOCAL_PROCESSOR_RECOVERED_IND`: The `SL_LOCAL_PROCESSOR_RECOVERED_IND` message is used by the SLS provider to indicate that a local processor outage condition no longer exists within the SLS provider.

A successful invocation of the local processor outage service is illustrated in Figure 3.26.

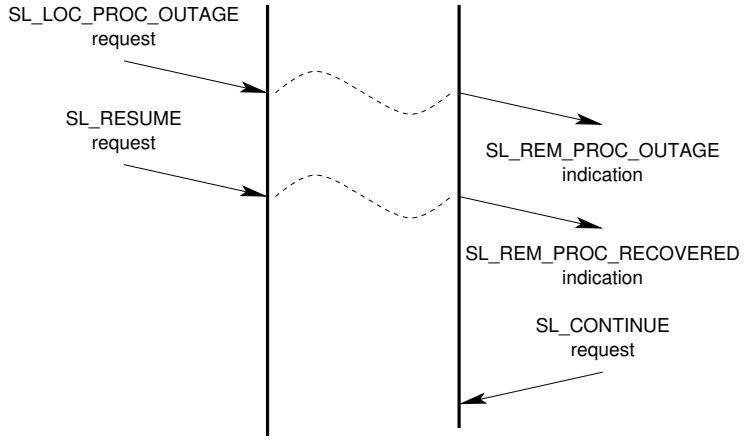

Figure 3.26: *Message Flow: Successful Processor Outage Service*

3.2.5.2 Remote Processor Outage Service

The remote processor outage service provides the SLS user with the ability to be informed of remote processor outage conditions. Remote processor outage occurs when the remote SLS user is experiencing a local processor outage. Remote processor outage conditions can exist independent of local processor outage conditions.

- `SL_REMOTE_PROCESSOR_OUTAGE_IND`: The `SL_REMOTE_PROCESSOR_OUTAGE_IND` message is used by the SLS provider to indicate that a remote processor outage condition exists.
- `SL_REMOTE_PROCESSOR_RECOVERED_IND`: The `SL_REMOTE_PROCESSOR_RECOVERED_IND` message is used by the SLS provider to indicate that a remote processor has recovered.
- `SL_CONTINUE_REQ`: The `SL_CONTINUE_REQ` message is used by the SLS user to request that a signalling link continue from where it left off after a remote processor has recovered.

A successful indication of the remote processor outage service is illustrated in Figure 3.27.

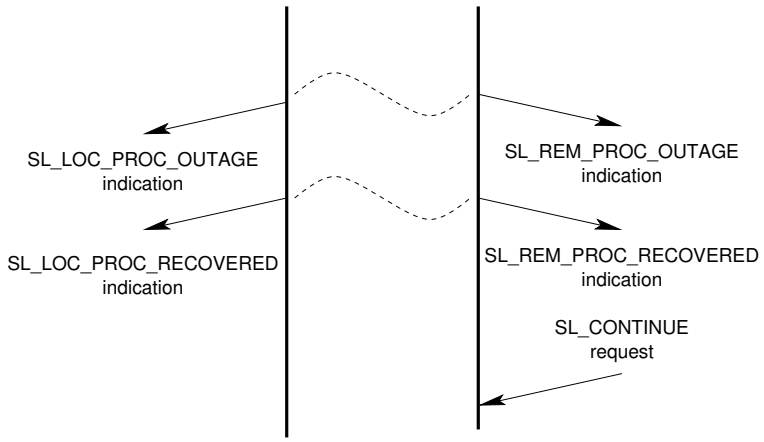

Figure 3.27: *Message Flow: Successful Processor Outage Service*

3.2.6 Link Option Management Service

The link option management service provides the SLS user with the ability to alter signalling link options. The service primitives that implement the link option management services are described in detail in Section 4.2.6 [Link Option Management Service Primitives], page 120.

- `SL_OPTMGMT_REQ`: The `SL_OPTMGMT_REQ` message is used by the SLS user to request that link options be managed.
- `SL_OPTMGMT_ACK`: The `SL_OPTMGMT_ACK` message is used by the SLS provider to acknowledge link option management actions.

A successful invocation of the link options management service is illustrated in Figure 3.28.

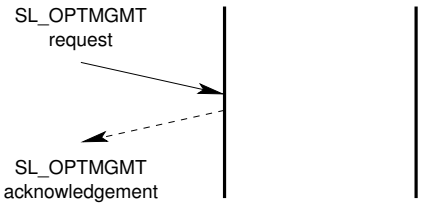

Figure 3.28: *Message Flow: Successful Link Options Management Service*

3.2.7 Event Notification Service

The event notification service provides the SLS user with the ability to register with the SLS provider to receive provider-specific event notifications. Event notifications normally correspond to management indications on the SS7 signalling link. The service primitives that implement the event notification services are described in detail in Section 4.2.7 [Event Notification Service Primitives], page 126.

- `SL_NOTIFY_REQ`: The `SL_NOTIFY_REQ` message is used by the SLS user to register with the SLS provider to receive specified event notifications.
- `SL_NOTIFY_IND`: The `SL_NOTIFY_IND` message is used by the SLS provider to indicate the occurrence of registered events to the SLS user.

A successful invocation of the event notification service is illustrated in Figure 3.29.

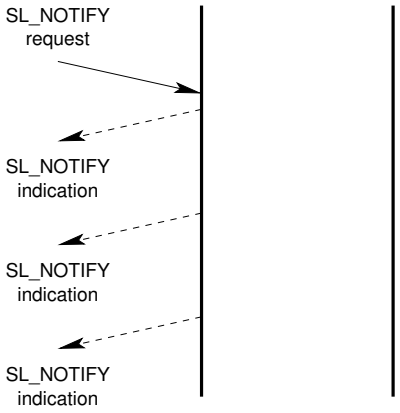

Figure 3.29: *Message Flow: Successful Event Notification Service*

4 SLI Primitives

4.1 Local Management Service Primitives

These service primitives implement the local management services (see Section 3.1 [Local Management Services], page 9).

4.1.1 Acknowledgement Service Primitives

These service primitives implement the acknowledgement service (see Section 3.1.1 [Acknowledgement Service], page 9).

4.1.1.1 LMI_OK_ACK

Description

This primitive is used to acknowledge receipt and successful service completion for primitives requiring acknowledgement that have no confirmation primitive.

Format

This primitive consists of one `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_long lmi_correct_primitive;
    lmi_ulong lmi_state;
} lmi_ok_ack_t;
```

Parameters

The service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_OK_ACK`.

`lmi_correct_primitive`
: Indicates the service primitive that was received and serviced correctly. This field can be one of the following values:

  `LMI_ATTACH_REQ`
  : Attach request.

  `LMI_DETACH_REQ`
  : Detach request.

`lmi_state`
: Indicates the current state of the LMS provider at the time that the primitive was issued. This field can be one of the following values:

  `LMI_UNATTACHED`
  : No PPA attached, awaiting `LMI_ATTACH_REQ`.

`LMI_UNUSABLE`
: Device cannot be used, STREAM in hung state.

`LMI_DISABLED`
: PPA attached, awaiting `LMI_ENABLE_REQ`.

`LMI_ENABLED`
: Ready for use, awaiting primitive exchange.

State

This primitive is issued by the LMS provider in the `LMI_ATTACH_PENDING` or `LMI_DETACH_PENDING` state.

New State

The new state is `LMI_UNATTACHED` or `LMI_DISABLED`, depending on thee primitive to which the message is responding.

4.1.1.2 LMI_ERROR_ACK

Description

The error acknowledgement primitive is used to acknowledge receipt and unsuccessful service completion for primitives requiring acknowledgement.

Format

The error acknowledgement primitive consists of one `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_errno;
    lmi_ulong lmi_reason;
    lmi_long lmi_error_primitive;
    lmi_ulong lmi_state;
} lmi_error_ack_t;
```

Parameters

The error acknowledgement primitive contains the following parameters:

`lmi_primitive`
: Indicates the primitive type. Always `LMI_ERROR_ACK`.

`lmi_errno`
: Indicates the LM error number. This field can have one of the following values:

  `LMI_UNSPEC`
  : Unknown or unspecified.

  `LMI_BADADDRESS`
  : Address was invalid.

  `LMI_BADADDRTYPE`
  : Invalid address type.

  `LMI_BADDIAL`
  : (Not used.)

  `LMI_BADDIALTYPE`
  : (Not used.)

  `LMI_BADDISPOSAL`
  : Invalid disposal parameter.

  `LMI_BADFRAME`
  : Defective SDU received.

  `LMI_BADPPA`
  : Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

`lmi_reason`
: Indicates the reason for failure. This field is protocol-specific. When the `lmi_errno` field is `LMI_SYSERR`, the `lmi_reason` field is the UNIX error number as described in `errno(3)`.

`lmi_error_primitive`
: Indicates the primitive that was in error. This field can have one of the following values:

`LMI_INFO_REQ`
: Information request.

`LMI_ATTACH_REQ`
: Attach request.

`LMI_DETACH_REQ`
: Detach request.

`LMI_ENABLE_REQ`
: Enable request.

`LMI_DISABLE_REQ`
: Disable request.

`LMI_OPTMGMT_REQ`
: Options management request.

`LMI_INFO_ACK`
: Information acknowledgement.

`LMI_OK_ACK`
: Successful receipt acknowledgement.

`LMI_ERROR_ACK`
: Error acknowledgement.

`LMI_ENABLE_CON`
: Enable confirmation.

`LMI_DISABLE_CON`
: Disable confirmation.

`LMI_OPTMGMT_ACK`
: Options Management acknowledgement.

`LMI_ERROR_IND`
: Error indication.

`LMI_STATS_IND`
: Statistics indication.

`LMI_EVENT_IND`
: Event indication.

`lmi_state`
: Indicates the state of the LMS provider at the time that the primitive was issued. This field can have one of the following values:

`LMI_UNATTACHED`
: No PPA attached, awaiting `LMI_ATTACH_REQ`.

`LMI_ATTACH_PENDING`
: Waiting for attach.

`LMI_UNUSABLE`
: Device cannot be used, STREAM in hung state.

`LMI_DISABLED`
: PPA attached, awaiting `LMI_ENABLE_REQ`.

`LMI_ENABLE_PENDING`
: Waiting to send `LMI_ENABLE_CON`.

`LMI_ENABLED`
: Ready for use, awaiting primitive exchange.

`LMI_DISABLE_PENDING`
: Waiting to send `LMI_DISABLE_CON`.

`LMI_DETACH_PENDING`
: Waiting for detach.

State

This primitive can be issued in any state for which a local acknowledgement is not pending. The LMS provider state at the time that the primitive was issued is indicated in the primitive.

New State

The new state remains unchanged.

4.1.2 Information Reporting Service Primitives

These service primitives implement the information reporting service (see Section 3.1.2 [Information Reporting Service], page 10).

4.1.2.1 LMI_INFO_REQ

Description

This LMS user originated primitive is issued by the LMS user to request that the LMS provider return information concerning the capabilities and state of the LMS provider.

Format

The primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_ulong lmi_primitive;
} lmi_info_req_t;
```

Parameters

This primitive contains the following parameters:

`lmi_primitive`
: Specifies the primitive type. Always `LMI_INFO_REQ`.

State

This primitive may be issued in any state but only when a local acknowledgement is not pending.

New State

The new state remains unchanged.

Response

This primitive requires the LMS provider to acknowledge receipt of the primitive as follows:

- **Successful**: The LMS provider is required to acknowledge receipt of the primitive and provide the requested information using the `LMI_INFO_ACK` primitive.
- **Unsuccessful (non-fatal errors)**: The LMS provider is required to negatively acknowledge the primitive using the `LMI_ERROR_ACK` primitive, and include the reason for failure in the primitive.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.2.2 LMI_INFO_ACK

Description

This LMS provider originated primitive acknowledges receipt and successful processing of the LMI_INFO_REQ primitive and provides the request information concerning the LMS provider.

Format

This message is formatted a one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_version;
    lmi_ulong lmi_state;
    lmi_ulong lmi_max_sdu;
    lmi_ulong lmi_min_sdu;
    lmi_ulong lmi_header_len;
    lmi_ulong lmi_ppa_style;
    lmi_uchar lmi_ppa_addr[0];
} lmi_info_ack_t;
```

Parameters

The information acknowledgement service primitive has the following parameters:

lmi_primitive
: Indicates the service primitive type. Always LMI_INFO_ACK.

lmi_version
: Indicates the version of this specification that is being used by the LMS provider.

lmi_state
: Indicates the state of the LMS provider at the time that the information acknowledgement service primitive was issued. This field can be one of the following values:

  LMI_UNATTACHED
  : No PPA attached, awaiting LMI_ATTACH_REQ.

  LMI_ATTACH_PENDING
  : Waiting for attach.

  LMI_UNUSABLE
  : Device cannot be used, STREAM in hung state.

  LMI_DISABLED
  : PPA attached, awaiting LMI_ENABLE_REQ.

  LMI_ENABLE_PENDING
  : Waiting to send LMI_ENABLE_CON.

`LMI_ENABLED`
: Ready for use, awaiting primitive exchange.

`LMI_DISABLE_PENDING`
: Waiting to send `LMI_DISABLE_CON`.

`LMI_DETACH_PENDING`
: Waiting for detach.

`lmi_max_sdu`
: Indicates the maximum size of a Service Data Unit.

`lmi_min_sdu`
: Indicates the minimum size of a Service Data Unit.

`lmi_header_len`
: Indicates the amount of header space that should be reserved for placing LMS provider headers.

`lmi_ppa_style`
: Indicates the PPA style of the LMS provider. This value can be one of the following values:

  `LMI_STYLE1`
  : PPA is implicitly attached by `open(2)`.

  `LMI_STYLE2`
  : PPA must be explicitly attached using `LMI_ATTACH_REQ`.

`lmi_ppa_addr`
: This is a variable length field. The length of the field is determined by the length of the `M_PROTO` or `M_PCPROTO` message block.

  For a *Style 2* driver, when `lmi_ppa_style` is `LMI_STYLE2`, and when in an attached state, this field providers the current PPA associated with the stream; the length is typically 4 bytes.

  For a *Style 1* driver, when `lmi_ppa_style` is `LMI_STYLE1`, the length it 0 bytes.

State

This primitive can be issued in any state where a local acknowledgement is not pending.

New State

The new state remains unchanged.

4.1.3 Physical Point of Attachment Service Primitives

These service primitives implement the physical point of attachment service (see Section 3.1.3 [Physical Point of Attachment Service], page 10).

4.1.3.1 LMI_ATTACH_REQ

Description

This LMS user originated primitive requests that the stream upon which the primitive is issued by associated with the specified Physical Point of Attachment (PPA). This primitive is only applicable to *Style 2* LMS provider streams, that is, streams that return `LMI_STYLE2` in the `lmi_ppa_style` field of the `LMI_INFO_ACK`.

Format

This primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_uchar lmi_ppa[0];
} lmi_attach_req_t;
```

Parameters

The attach request primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `LMI_ATTACH_REQ`.

`lmi_ppa`
: Specifies the Physical Point of Attachment (PPA) to which to associated the *Style 2* stream. This is a variable length identifier whose length is determined by the length of the `M_PROTO` message block.

State

This primitive is only valid in state `LMI_UNATTACHED` and when a local acknowledgement is not pending.

New State

Upon success, the new state is `LMI_ATTACH_PENDING`. Upon failure, the state remains unchanged.

Response

The attach request service primitive requires that the LMS provider respond as follows:

- **Successful**: The LMS provider acknowledges receipt of the primitive and successful outcome of the attach service with a `LMI_OK_ACK` primitive. The new state is `LMI_DISABLED`.
- **Unsuccessful (non-fatal errors)**: The LMS provider acknowledges receipt of the primitive and failure of the attach service with a `LMI_ERROR_ACK` primitive containing the reason for failure. The new state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.3.2 LMI_DETACH_REQ

Description

This LMS user originated primitive request that the stream upon which the primitive is issued be disassociated from the Physical Point of Appearance (PPA) to which it is currently attached. This primitive is only applicable to *Style 2* LMS provider streams, that is, streams that return `LMI_STYLE2` in the `lmi_ppa_style` field of the `LMI_INFO_ACK`.

Format

The detach request service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
} lmi_detach_req_t;
```

Parameters

The detach request service primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `LMI_DETACH_REQ`.

State

This primitive is valid in the `LMI_DISABLED` state and when no local acknowledgement is pending.

New State

Upon success, the new state is `LMI_DETACH_PENDING`. Upon failure, the state remains unchanged.

Response

The detach request service primitive requires that the LMS provider respond as follows:

- **Successful**: The LMS provider acknowledges receipt of the primitive and successful outcome of the detach service with a `LMI_OK_ACK` primitive. The new state is `LMI_UNATTACHED`.
- **Unsuccessful (non-fatal errors)**: The LMS provider acknowledges receipt of the primitive and failure of the detach service with a `LMI_ERROR_ACK` primitive containing the reason for failure. The new state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.4 Initialization Service Primitives

Initialization service primitives allow the LMS user to enable or disable the protocol service interface. Enabling the protocol service interface may require that some action be taken to prepare the protocol service interface for use or to remove it from use. For example, where the PPA corresponds to a signalling data link identifier as defined in Q.704, it may be necessary to perform switching to connect or disconnect the circuit identification code associated with the signalling data link identifier.

These service primitives implement the initialization service (see Section 3.1.4 [Initialization Service], page 12).

4.1.4.1 LMI_ENABLE_REQ

Description

This LMS user originated primitive request that the LMS provider perform the actions necessary to enable the protocol service interface and confirm that it is enabled. This primitive is applicable to both styles of PPA.

Format

The enable request service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_uchar lmi_rem[0];
} lmi_enable_req_t;
```

Parameters

The enable request service primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `LMI_ENABLE_REQ`.

`lmi_rem`
: Specifies a remote address to which to connect the PPA. The need for and form of this address is provider-specific. The length of the field is determined by the length of the `M_PROTO` message block. This remote address could be a circuit identification code, an IP address, or some other form of circuit or channel identifier.

State

This primitive is valid in the `LMI_DISABLED` state and when no local acknowledgement is pending.

New State

Upon success the new state is `LMI_ENABLE_PENDING`. Upon failure, the state remains unchanged.

Response

The enable request service primitive requires that the LMS provider acknowledge receipt of the primitive as follows:

- **Successful**: When successful, the LMS provider acknowledges successful completion of the enable service with an `LMI_ENABLE_CON` primitive. The new state is `LMI_ENABLED`.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the LMS provider acknowledges the failure of the enable service wtih an `LMI_ERROR_ACK` primitive containing the error. The new state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.4.2 LMI_ENABLE_CON

Description

This LMS provider originated primitive is issued by the LMS provider to confirm the successful completion of the enable service.

Format

The enable confirmation service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_state;
} lmi_enable_con_t;
```

Parameters

The enable confirmation service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_ENABLE_CON`.

`lmi_state`
: Indicates the state following issuing the enable confirmation primitive. This field can take on one of the following values:

  `LMI_ENABLED`
  : Ready for use, awaiting primitive exchange.

State

This primitive is issued by the LMS provider in the `LMI_ENABLE_PENDING` state.

New State

The new state is `LMI_ENABLED`.

4.1.4.3 LMI_DISABLE_REQ

Description

This LMS user originated primitive requests that the LMS provider perform the actions necessary to disable the protocol service interface and confirm that it is disabled. The primitive is applicable to both styles of PPA.

Format

The disable request service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
} lmi_disable_req_t;
```

Parameters

The disable request service primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `LMI_DISABLE_REQ`.

State

The disable request service primitive is valid in the `LMI_ENABLED` state and when no local acknowledgement is pending.

New State

Upon success, the new state is `LMI_DISABLE_PENDING`. Upon failure, the state remains unchanged.

Response

The disable request service primitive requires the LMS provider to acknowledge receipt of the primitive as follows:

- **Successful**: When successful, the LMS provider acknowledges successful completion of the disable service with an `LMI_DISABLE_CON` primitive. The new state is `LMI_DISABLED`.
- **Unsuccessful (non-fatal errors)**: When unsuccessful, the LMS provider acknowledges the failure of the disable service with an `LMI_ERROR_ACK` primitive containing the error. The new state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.4.4 LMI_DISABLE_CON

Description

This LMS provider originated primitive is issued by the LMS provider to confirm the successful completion of the disable service.

Format

The disable confirmation service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_state;
} lmi_disable_con_t;
```

Parameters

The disable confirmation service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_DISABLE_CON`.

`lmi_state`
: Indicates the state following issuing the disable confirmation primitive. This field can take on one of the following values:

  `LMI_DISABLED`
  : PPA attached, awaiting `LMI_ENABLE_REQ`.

State

This primitive is issued by the LMS provider in the `LMI_DISABLE_PENDING` state.

New State

The new state is `LMI_DISABLED`.

4.1.5 Options Management Service Primitives

The options management service primitives allow the LMS user to negotiate options with the LMS provider, retrieve the current and default values of options, and check that values specified for options are correct.

The options management service primitive implement the options management service (see Section 3.1.5 [Options Management Service], page 13).

4.1.5.1 LMI_OPTMGMT_REQ

Description

This LMS user originated primitive requests that LMS provider options be managed.

Format

The option management request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_opt_length;
    lmi_ulong lmi_opt_offset;
    lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_req_t;
```

Parameters

The option management request service primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `LMI_OPTMGMT_REQ`.

`lmi_opt_length`
: Specifies the length of the options.

`lmi_opt_offset`
: Specifies the offset, from the beginning of the `M_PROTO` message block, of the start of the options.

`lmi_mgmt_flags`
: Specifies the management flags which determine what operation the LMS provider is expected to perform on the specified options. This field can assume one of the following values:

  `LMI_NEGOTIATE`
  : Negotiate the specified value of each specified option and return the negotiated value.

  `LMI_CHECK`
  : Check the validity of the specified value of each specified option and return the result. Do not alter the current value assumed by the LMS provider.

`LMI_DEFAULT`
: Return the default value for the specified options (or all options). Do not alter the current value assumed by the LMS provider.

`LMI_CURRENT`
: Return the current value for the specified options (or all options). Do not alter the current value assumed by the LMS provider.

State

This primitive is valid in any state where a local acknowledgement is not pending.

New State

The new state remains unchanged.

Response

The option management request service primitive requires the LMS provider to acknowledge receipt of the primitive as follows:

- **Successful**: Upon success, the LMS provider acknowledges receipt of the service primitive and successful completion of the options management service with an `LMI_OPTMGMT_ACK` primitive containing the options management result. The state remains unchanged.
- **Unsuccessful (non-fatal errors)**: Upon failure, the LMS provider acknowledges receipt of the service primitive and failure to complete the options management service with an `LMI_ERROR_ACK` primitive containing the error. The state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.1.5.2 LMI_OPTMGMT_ACK

Description

This LMS provider originated primitive is issued by the LMS provider upon successful completion of the options management service. It indicates the outcome of the options management operation requested by the LMS user in a `LMI_OPTMGMT_REQ` primitive.

Format

The option management acknowledgement service primitive consists of one `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_opt_length;
    lmi_ulong lmi_opt_offset;
    lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_ack_t;
```

Parameters

The option management acknowledgement service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_OPTMGMT_ACK`.

`lmi_opt_length`
: Indicates the length of the returned options.

`lmi_opt_offset`
: Indicates the offset of the returned options from the start of the `M_PCPROTO` message block.

`lmi_mgmt_flags`
: Indicates the returned management flags. These flags indicate the overall success of the options management service. This field can assume one of the following values:

  `LMI_SUCCESS`
  : The LMS provider succeeded in negotiating or returning all of the options specified by the LMS user in the `LMI_OPTMGMT_REQ` primitive.

  `LMI_FAILURE`
  : The LMS provider failed to negotiate one or more of the options specified by the LMS user.

  `LMI_PARTSUCCESS`
  : The LMS provider negotiated a value of lower quality for one or more of the options specified by the LMS user.

`LMI_READONLY`
: The LMS provider failed to negotiate one ore more of the options specified by the LMS user because the option is treated as read-only by the LMS provider.

`LMI_NOTSUPPORT`
: The LMS provider failed to recognize one or more of the options specified by the LMS user.

State

This primitive is issued by the LMS provider in direct response to an `LMI_OPTMGMT_REQ` primitive.

New State

The new state remains unchanged.

Rules

The LMS provider follows the following rules when processing option management service requests:

— When the `lmi_mgmt_flags` field in the `LMI_OPTMGMT_REQ` primitive is set to `LMI_NEGOTIATE`, the LMS provider will attempt to negotiate a value for each of the options specified in the request.

— When the flags are `LMI_DEFAULT`, the LMS provider will return the default values of the specified options, or the default values of all options known to the LMS provider if no options were specified.

— When the flags are `LMI_CURRENT`, the LMS provider will return the current values of the specified options, or all options.

— When the flags are `LMI_CHECK`, the LMS provider will attempt to negotiate a value for each of the options specified in the request and return the resulg of the negotiation, but will not affect the current value of the option.

4.1.6 Event Reporting Service Primitives

The event reporting service primitives allow the LMS provider to indicate asynchronous errors, events and statistics collection to the LMS user.

These service primitives implement the event reporting service (see Section 3.1.8 [Event Reporting Service], page 15).

4.1.6.1 LMI_ERROR_IND

Description

This LMS provider originated service primitive is issued by the LMS provider when it detects and asynchronous error event. The service primitive is applicable to all styles of PPA.

Format

The error indication service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_errno;
    lmi_ulong lmi_reason;
    lmi_ulong lmi_state;
} lmi_error_ind_t;
```

Parameters

The error indication service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_ERROR_IND`.

`lmi_errno`
: Indicates the LMI error number describing the error. This field can have one of the following values:

  `LMI_UNSPEC`
  : Unknown or unspecified.

  `LMI_BADADDRESS`
  : Address was invalid.

  `LMI_BADADDRTYPE`
  : Invalid address type.

  `LMI_BADDIAL`
  : (Not used.)

  `LMI_BADDIALTYPE`
  : (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

`lmi_reason`
: Indicates the reason for failure. This field is protocol-specific. When the `lmi_errno` field is `LMI_SYSERR`, the `lmi_reason` field is the UNIX error number as described in `errno(3)`.

`lmi_state`
: Indicates the state of the LMS provider at the time that the primitive was issued. This field can have one of the following values:

`LMI_UNATTACHED`
: No PPA attached, awaiting `LMI_ATTACH_REQ`.

`LMI_ATTACH_PENDING`
: Waiting for attach.

`LMI_UNUSABLE`
: Device cannot be used, STREAM in hung state.

`LMI_DISABLED`
: PPA attached, awaiting `LMI_ENABLE_REQ`.

`LMI_ENABLE_PENDING`
: Waiting to send `LMI_ENABLE_CON`.

`LMI_ENABLED`
: Ready for use, awaiting primitive exchange.

`LMI_DISABLE_PENDING`
: Waiting to send `LMI_DISABLE_CON`.

`LMI_DETACH_PENDING`
: Waiting for detach.

State

This primitive can be issued in any state for which a local acknowledgement is not pending. The LMS provider state at the time that the primitive was issued is indicated in the primitive.

New State

The new state remains unchanged.

4.1.6.2 LMI_STATS_IND

Description

This LMS provider originated primitive is issued by the LMS provider to indicate a periodic statistics collection event. The service primitive is applicable to all styles of PPA.

Format

The statistics indication service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_interval;
    lmi_ulong lmi_timestamp;
} lmi_stats_ind_t;
```

Following this structure within the `M_PROTO` message block is the provider-specific statistics.

Parameters

The statistics indication service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_STATS_IND`.

`lmi_interval`
: Indicates the statistics collection interval to which the statistics apply. This interval is specified in milliseconds.

`lmi_timestamp`
: Indicates the UNIX time (from epoch) at which statistics were collected. The timestamp is given in milliseconds from epoch.

State

This service primitive may be issued by the LMS provider in any state in which a local acknowledgement is not pending.

New State

The new state remains unchanged.

4.1.6.3 LMI_EVENT_IND

Description

This LMS provider originated primitive is issued by the LMS provider to indicate an asynchronous event. The service primitive is applicable to all styles of PPA.

Format

The event indication service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_objectid;
    lmi_ulong lmi_timestamp;
    lmi_ulong lmi_severity;
} lmi_event_ind_t;
```

Following this structure within the `M_PROTO` message block is the provider-specific event information.

Parameters

THe event indication service primitive contains the following parameters:

`lmi_primitive`
: Indicates the service primitive type. Always `LMI_EVENT_IND`.

`lmi_objectid`
: Indicates the provider-specific object identifier that identifies the managed object to which the event is associated.

`lmi_timestamp`
: Indicates the UNIX time from epoch (in milliseconds).

`lmi_severity`
: Indicates the provider-specific severity of the event.

State

This service primitive can be issued by the LMS provider in any state where a local acknowledgement is not pending. Normally the LMS provider must be in the `LMI_ENABLED` state for event reporting to occur.

New State

The new state remains unchanged.

4.2 Protocol Service Primitives

Protocol service primitives implement the Signalling Link interface protocol. Protocol service primitives provide the SLS user with the ability to initialize the link, transfer data on the link, request and receive reports of receive and transmit congestion, restore failed signalling links, handle processor outage conditions, manage options and register for and receive event notifications.

These service primitives implement the protocol services (see Section 3.2 [Protocol Services], page 15).

4.2.1 Link Initialization Service Primitives

The link initialization primitives permit the SLS user to power on the signalling data terminal, specify emergency or normal alignment, start the signalling link and bring it into service, and stop the signalling link or be informed of link failures.

These service primitives implement the link initialization services (see Section 3.2.1 [Link Initialization Services], page 15).

4.2.1.1 SL_POWER_ON_REQ

Description

The SLS user originated service primitive request that the SLS provider power on the signalling data terminal. Not all signalling data terminals can be powered on independent of the existence of the signalling link interface. Software signalling data terminals will mark idle on signalling links until they are powered on, after which they will idle FISUs.

Format

The power on service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_power_on_req_t;
```

Parameters

The power on service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_POWER_ON_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state. This primitive is valid in the `SL_STATE_POWER_OFF` link state; however, when issued in another link state the primitive is ignored and does not generate a non-fatal error.

New State

The new link state is `SL_STATE_OUT_OF_SERVICE`.

Rules

Response

The power on service primitive does not require receipt acknowledgement from the SLS provider.

- **Successful:** When successful, the power on service primitive does not require acknowledgement.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider indicates failure using an `LMI_ERROR_ACK` primitive containing the error.

Note that the SLS provider should ignore this primitive, and not generate a non-fatal error, when the management interface is in the `LMI_ENABLED` state and the link state is other than `SL_STATE_POWER_OFF`.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.1.2 SL_EMERGENCY_REQ

Description

The emergency request service primitive provides the SLS user with the ability to specify that emergency alignment procedures should be used on the current or next alignment of the signalling link. Emergency alignment procedures a shorter in duration (shorter proving period) than normal alignment procedures.

Format

The emergency request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_emergency_req_t;
```

Parameters

The emergency request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_EMERGENCY_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state. The primitive is valid in any link state.

New State

The management and link state remains unchanged.

Response

The emergency request service primitive does not require receipt acknowledgement.

- **Successful:** When successful, the emergency request service primitive does not require receipt acknowledgement.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive with an `LMI_ERROR_ACK` primitive containing the error.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.1.3 SL_EMERGENCY_CEASES_REQ

Description

The emergency ceases request service primitive provides the SLS user with the ability to specify that normal alignment procedures should be used on the current or next alignment of the signalling link. Normal alignment procedures are longer in duration (longer proving period) than emergency alignment procedures.

Format

The emergency ceases request primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_emergency_ceases_req_t;
```

Parameters

The emergency ceases request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_EMERGENCY_CEASES_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state. The primitive is valid in any link state.

New State

The management and link state remains unchanged.

Response

The emergency ceases request service primitive does not require receipt acknowledgement.

- **Successful:** When successful, the emergency ceases request service primitive does not require receipt acknowledgement.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive with an `LMI_ERROR_ACK` primitive containing the error.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.1.4 SL_START_REQ

Description

The start request service primitive allows the SLS user to request that a signalling link be aligned and brought into service by the SLS provider.

Format

The start request service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_start_req_t;
```

Parameters

The start request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_START_REQ.

State

This primitive is only valid in management state LMI_ENABLED. This primitive is valid in link state SL_STATE_OUT_OF_SERVICE.

New State

The new link state is SL_STATE_INITIAL_ALIGNMENT.

Response

The start request service primitive requires a response from the SLS provider indicating the success or failure of the start request.

- **Successful link start:** When successful, the SLS provider indicates success with the SL_IN_SERVICE_IND primitive indicating that the signalling link has been brought into service. A significant delay in time might exist between the request and the in-service indication. This results in the SL_STATE_IN_SERVICE link state.
- **Unuccessful link start:** When unsuccessful, the SLS provider indicates failure to bring the link in-service with the SL_OUT_OF_SERVICE_IND primitive, containing the reason for failure. This results int the SL_STATE_OUT_OF_SERVICE link state.
- **Non-fatal errors:** Non-fatal errors are indicated by the SLS provider using the LMI_ERROR_ACK primitive with the error number and reason contained.

When the management state is LMI_ENABLED, but the link state is other than SL_STATE_OUT_OF_SERVICE and SL_STATEPOWER_OFF, the SLS provider should ignore the SL_START_REQ primitive and not generate a non-fatal error.

Reason for Failure

Applicable reasons for unsuccessful link start are as follows:

`SL_FAIL_UNSPECIFIED`
: The signalling link failed for an unspecified reason.

`SL_FAIL_CONG_TIMEOUT`
: The signalling link failed because of congestion timeout (T6 expiry).

`SL_FAIL_ACK_TIMEOUT`
: The signalling link failed because of acknowledgement timeout (T7 expiry).

`SL_FAIL_ABNORMAL_BSNR`
: The signalling link failed because of receipt of an abnormal backward sequence number (BSNR).

`SL_FAIL_ABNORMAL_FIBR`
: The signalling link failed because of receipt of an abnormal forward indicator bit (FIBR).

`SL_FAIL_SUERM_EIM`
: The signalling link failed because the SUERM or EIM error rate threshold was exceeded.

`SL_FAIL_ALIGNMENT_NOT_POSSIBLE`
: The signalling link failed because the AERM threshold was exceeded and the maximum number of proving periods was exceeded.

`SL_FAIL_RECEIVED_SIO`
: The signalling link failed due to receipt of an SIO during or after alignment.

`SL_FAIL_RECEIVED_SIN`
: The signalling link failed due to receipt of an SIN after proving.

`SL_FAIL_RECEIVED_SIE`
: The signalling link failed due to receipt of an SIE after proving.

`SL_FAIL_RECEIVED_SIOS`
: The signalling link failed due to receipt of an SIOS.

`SL_FAIL_T1_TIMEOUT`
: The signalling link failed due to failure to align with remote (T1 timeout).

Applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.1.5 SL_IN_SERVICE_IND

Description

The in-service indication service primitive is issued by the SLS provider to indicate to the SLS user that a previously invoked link start has successfully aligned and brought the signalling link into service.

Format

The in-service indication service primitive consists of one M_PROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_in_service_ind_t;
```

Parameters

The in-service indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_IN_SERVICE_IND.

State

This primitive is only issued in the LMI_ENABLED management state. This primitive is only issued in the SL_STATE_ALIGNED_READY state.

New State

The new link state is SL_STATE_IN_SERVICE.

Rules

The following rules are observed by the SLS provider when issuing the in-service indication primitive:

— The primitive is only issued in response to a SL_START_REQ primitive that was issued from the SL_STATE_OUT_OF_SERVICE state.

— The primitive is only issued once the signalling link has achieved the SL_STATE_IN_SERVICE state.

4.2.1.6 SL_OUT_OF_SERVICE_IND

Description

The out-of-service indication service primitive is issued by the SLS provider to indicate to the SLS user that a previously invoked link start has been unsuccessful, or that a previously in-service signalling link has failed.

Format

The out-of-service indication service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
    sl_ulong sl_reason;
} sl_out_of_service_ind_t;
```

Parameters

The out-of-service indication service primitive contains the following parameters:

`sl_primitive`
: Indicates the service primitive type. Always `SL_OUT_OF_SERVICE_IND`.

`sl_timestamp`
: Indication the time of the failure. The time is indicated as UNIX time from epoch in milliseconds.

`sl_reason`
: Indicates the reason for failure to start the link or the reason for failure of an in-service link. The `sl_reason` field can assume one of the following values:

  `SL_FAIL_UNSPECIFIED`
  : The signalling link failed for an unspecified reason.

  `SL_FAIL_CONG_TIMEOUT`
  : The signalling link failed because of congestion timeout (T6 expiry).

  `SL_FAIL_ACK_TIMEOUT`
  : The signalling link failed because of acknowledgement timeout (T7 expiry).

  `SL_FAIL_ABNORMAL_BSNR`
  : The signalling link failed because of receipt of an abnormal backward sequence number (BSNR).

  `SL_FAIL_ABNORMAL_FIBR`
  : The signalling link failed because of receipt of an abnormal forward indicator bit (FIBR).

  `SL_FAIL_SUERM_EIM`
  : The signalling link failed because the SUERM or EIM error rate threshold was exceeded.

`SL_FAIL_ALIGNMENT_NOT_POSSIBLE`
: The signalling link failed because the AERM threshold was exceeded and the maximum number of proving periods was exceeded.

`SL_FAIL_RECEIVED_SIO`
: The signalling link failed due to receipt of an SIO during or after alignment.

`SL_FAIL_RECEIVED_SIN`
: The signalling link failed due to receipt of an SIN after proving.

`SL_FAIL_RECEIVED_SIE`
: The signalling link failed due to receipt of an SIE after proving.

`SL_FAIL_RECEIVED_SIOS`
: The signalling link failed due to receipt of an SIOS.

`SL_FAIL_T1_TIMEOUT`
: The signalling link failed due to failure to align with remote (T1 timeout).

State

This primitive is only issued in the `LMI_ENABLED` management state. This primitive is only issued from a link state other than `SL_STATE_OUT_OF_SERVICE` or `SL_STATE_POWER_OFF`.

New State

The new link state is `SL_STATE_OUT_OF_SERVICE`.

Rules

The following rules are observed by the SLS provider when issuing the out-of-service indication primitive:

— The primitive is only issued in response to a `SL_START_REQ` primitive that was issued from the `SL_STATE_OUT_OF_SERVICE` state, or as a result of a link failure from the `SL_STATE_IN_SERVICE` state.

— The primitive is only issued once the signalling link has achieved the `SL_STATE_OUT_OF_SERVICE` state.

4.2.1.7 SL_STOP_REQ

Description

The stop request primitive allows the SLS user to request that a signalling link be brought out of service by the SLS provider.

Format

The stop request service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_stop_req_t;
```

Parameters

The stop request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_STOP_REQ.

State

This primitive is only valid in management state LMI_ENABLED. This primitive is valid in link state SL_STATE_INITIAL_ALIGNMENT, SL_STATE_ALIGNED_READY, SL_STATE_ALIGNED_NOT_READY or SL_STATE_IN_SERVICE.

New State

The new link state is SL_STATE_OUT_OF_SERVICE.

Response

The stop request service primitive does not require receipt acknowledgement from the SLS provider.

- **Successful:** When successful, the SLS provider does not need to acknowledge the stop request service primitive. The resulting link state is SL_STATE_OUT_OF_SERVICE.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the stop request service primitive with a LMI_ERROR_ACK primitive containing the error and reason. The resulting state is unchanged.

When the management state is LMI_ENABLED, but the link state is SL_STATE_POWER_OFF or SL_STATE_OUT_OF_SERVICE, the SLS provider should ignore the SL_STOP_REQ primitive and not generate a non-fatal error.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

LMI_UNSPEC
: Unknown or unspecified.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.2 Data Transfer Service Primitives

Data transfer service primitives provider the SLS user with the ability to send and receive message signal units on an in-service signalling link. These service primitives implement the data transfer service (see Section 3.2.2 [Data Transfer Service], page 18).

4.2.2.1 SL_PDU_REQ

Description

The PDU request service primitive provides the SLS user with the ability to request that a message signal unit be transmitted on an in-service signalling link.

Format

The PDU request service primitive consists of zero or one `M_PROTO` message block and one `M_DATA` message block containing the message signal unit. The structure of the `M_PROTO` message block is as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_mp;
} sl_pdu_req_t;
```

Parameters

The PDU request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_PDU_REQ`.

`sl_mp`
: Specifies the message priority for the message signal unit. Message priorities are provider-specific, but are typically between 0 and 3. This message priority field is only applicable to SS7 protocol variants that place message priority bits in a field of the Level 2 header (TTC).

State

This primitive is only valid in the `LMI_ENABLED` management state, and is valid from the `SL_STATE_IN_SERVICE` link state.

New State

The management and link state remains unchanged.

Rules

The following rules are observed when issuing the PDU request service primitive:

— The `M_PROTO` message block is optional and is only necessary for the TTC SS7 protocol variant, or an SS7 protocol variant which places message priority bits into the Level 2 header.

— The PDU request service primitive does not require a response from the SLS provider.

Response

The PDU request service primitive is not acknowledged.

4.2.2.2 SL_PDU_IND

Description

The PDU indication service primitive provides the SLS user with the ability to receive message signal units from a signalling link.

Format

The PDU indication service primitive consists of zero or more `M_PROTO` message blocks and one or more `M_DATA` message blocks containing the message signal unit. The structure of the `M_PROTO` message block is as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_mp;
} sl_pdu_ind_t;
```

Parameters

The PDU indication service primitive contains the following parameters:

`sl_primitive`
: Indicates the service primitive type. Always `SL_PDU_IND`.

`sl_mp`
: Indicates the message priority of the message signal unit. Message priorities are provider-specific, but are typically between 0 and 3. This message priority field is only applicable to SS7 protocol variants that place message priority bits in a field of the Level 2 header (e.g. TTC).

State

This primitive is only valid in the `LMI_ENABLED` management state, and is valid from the `SL_STATE_IN_SERVICE` link state.

New State

The management and link states remain unchanged.

Rules

The following rules are observed when issuing the PDU indication service primitive:

— The `M_PROTO` message block is optional and is only necessary for the TTC SS7 protocol variant, or an SS7 protocol variant that passes message priority bits from the Level 2 header.

— The PDU indication service primitive does not require a response from the SLS user.

4.2.3 Congestion Service Primitives

These service primitives implement the congestion services (see Section 3.2.3 [Congestion Services], page 18).

4.2.3.1 SL_LINK_CONGESTED_IND

Description

The link congested indication service primitive provides the SLS provider with the ability to inidicate link transmit congestion onset at a congestion level to the SLS user.

Format

The link congested indication service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
    sl_ulong sl_cong_status;            /* congestion status */
    sl_ulong sl_disc_status;            /* discard status */
} sl_link_cong_ind_t;
```

Parameters

The link congested indication service primitive contains the following parameters:

`sl_primitive`
: Indicates the service primitive type. Always `SL_LINK_CONGESTED_IND`.

`sl_timestamp`
: Indicates the time at which the change in congestion status occurred. This is UNIX time from epoch timestamp in milliseconds.

`sl_cong_status`
: Indicates the congestion status. The congestion status is the maximum level at which transmit congestion onset has occurred. This field is provider-specific but can typically take on a value from 0 to 3. The SLS user should signal congestion to the senders of messages with message priority less than the congestion status but should not discard messages of that priority.

`sl_disc_status`
: Indicates the discard status. The discard status is the maximum level at which transmit congestion discard has occurred. This field is provider-specific but can typically take on a value from 0 to 3. The SLS user should signal congestion to senders of message with message priority less than the discard status and should also discard messages of that priority.

State

This primitive is only issued in the `LMI_ENABLED` management state and the `SL_STATE_IN_SERVICE` link state.

New State

The management and link state remain unchanged.

Rules

The SLS provider observes the following rules when issuing the link congested indication service primitive:

— The service primitive is only issued from the `SL_STATE_IN_SERVICE` link state.

— The service primitive is only issued from the `LMI_ENABLED` management state.

— The service primitive is only issued when the congestion status or discard status increases from the value that was last indicated with either a `SL_LINK_CONGESTION_IND` or `SL_LINK_CONGESTION_CEASED_IND` primitive.

Response

The SLS user upon receiving this primitive should avoid sending messages of message priority less than the transmit congestion status, and must not send messages of message priority less than the discard status. The SLS provider does not actually discard messages with message priority less than the discard status: it is the responsibility of the SLS user to discard lower priority messages.

Typically the SLS user is the SS7 Message Transfer Part. The SS7 MTP issues congestion indications to local MTP-Users and issues transfer-controlled messages to sending signalling points when transmit congestion onset occurs. When transmit congestion discard occurs, the SS7 MTP continues to issue congestion indications to local MTP-User and transfer-controlled message to sending signalling points, but also discards messages with insufficient priority for the discard level.

4.2.3.2 SL_LINK_CONGESTION_CEASED_IND

Description

The link congestion ceased indication service primitive allows the SLS provider to indicate to the SLS user when transmit congestion abates.

Format

The link congestion ceased service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
    sl_ulong sl_cong_status;            /* congestion status */
    sl_ulong sl_disc_status;            /* discard status */
} sl_link_cong_ceased_ind_t;
```

Parameters

The link congestion ceased service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_CONGESTION_CEASED_IND.

sl_timestamp
: Indicates the time at which the change in transmit congestion status occurred. This is UNIX time from epoch timestamp in milliseconds.

sl_cong_status
: Indicates the congestion status. The congestion status is the maximum level at which transmit congestion onset has occurred. This field is provider-specific but can typically take on a value from 0 to 3. The SLS user should signal congestion to the senders of messages with message priority less than the congestion status but should not discard messages of that priority.

sl_disc_status
: Indicates the discard status. The discard status is the maximum level at which transmit congestion discard has occurred. This field is provider-specific but can typically take on a value from 0 to 3. The SLS user should signal congestion to senders of message with message priority less than the discard status and should also discard messages of that priority.

State

This primitive is only issued in the LMI_ENABLED management state and the SL_STATE_IN_SERVICE link state.

New State

The management and link state remain unchanged.

Rules

The SLS provider observes the following rules when issuing the link congestion ceased indication service primitive:

— The service primitive is only issued from the `SL_STATE_IN_SERVICE` link state.

— The service primitive is only issued from the `LMI_ENABLED` management state.

— The service primitive is only issued when the congestion status or discard status decreases from the value that was last indicated with either a `SL_LINK_CONGESTION_IND` or `SL_LINK_CONGESTION_CEASED_IND` primitive.

Response

The SLS user upon receiving this primitive should cease discarding or sending congestion indications or transfer-controlled messages for the congestion level which has abated.

4.2.3.3 SL_CONGESTION_DISCARD_REQ

Description

The congestion discard request service primitive is used by the SLS user to specify receive congestion discard.

Normally an SLS user will first signal receive congestion onset with the `SL_CONGESTION_ACCEPT_REQ` primitive before signalling receive congestion discard with this `SL_CONGESTION_DISCARD_REQ` primitive. The congestion discard service primitive requests that the SLS provider discard all new undelivered message signal units and not acknowledge them to the remote SLS provider. The SLS provider will also generate receive congestion indications to the remote SLS provider (i.e. will periodically generate SIB).

Format

The congestion discard request service primitive consists of one `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_cong_discard_req_t;
```

Parameters

The congestion discard request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_CONGESTION_DISCARD_REQ`.

State

This primitive is valid only in `LMI_ENABLED` management state. It is valid in `SL_STATE_IN_SERVICE` link state.

New State

The link and management state remains the same.

Rules

The SLS user should observe the following rules when issuing the congestion discard request service primitive:

— The SLS user should not generate a congestion discard request unless a congestion accept request was previously issued.

— The SLS user should not generate a congestion discard request unless a congestion accept request was previously issued *and* a message signal unit has been delivered since the congestion accept request was issued.

Response

The congestion discard request service primitive does not require receipt acknowledgement.

- **Successful:** When successful, this primitive does not require acknowledgement. The state remains the same.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using the `LMI_ERROR_ACK` primitive containing the error and reason. The state remains the same.

Note that if the SLS provider is in the `LMI_ENABLED` state, but the link is not in the `SL_STATE_IN_SERVICE` state, the primitive should be ignored and no non-fatal error generated.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.3.4 SL_CONGESTION_ACCEPT_REQ

Description

The congestion accept request service primitive is used by the SLS user to specify receive congestion onset.

Format

The congestion accept request service primitive consists of one M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_cong_accept_req_t;
```

Parameters

The congestion accept request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_CONGESTION_ACCEPT_REQ.

State

This primitive is valid only in LMI_ENABLED management state. It is valid in SL_STATE_IN_SERVICE link state.

New State

The link and management state remains the same.

Response

The congestion accept request service primitive does not require receipt acknowledgement.

- **Successful:** When successful, this primitive does not require acknowledgement. The state remains the same.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using the LMI_ERROR_ACK primitive containing the error and reason. The state remains the same.

Note that if the SLS provider is in the LMI_ENABLED state, but the link is not in the SL_STATE_IN_SERVICE state, the primitive should be ignored and no non-fatal error generated.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

LMI_UNSPEC
: Unknown or unspecified.

LMI_EVENT
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.3.5 SL_NO_CONGESTION_REQ

Description

The no congestion request service primitive is used by the SLS user to specify receive congestion abatement.

Format

The no congestion request service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_no_cong_req_t;
```

Parameters

The no congestion request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_NO_CONGESTION_REQ.

State

This primitive is valid only in LMI_ENABLED management state. It is valid in SL_STATE_IN_SERVICE link state.

New State

The link and management state remains the same.

Response

The no congestion request service primitive does not require receipt acknowledgement.

- **Successful:** When successful, this primitive does not require acknowledgement. The state remains the same.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using the LMI_ERROR_ACK primitive containing the error and reason. The state remains the same.

Note that if the SLS provider is in the LMI_ENABLED state, but the link is not in the SL_STATE_IN_SERVICE state, the primitive should be ignored and no non-fatal error generated.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

LMI_UNSPEC
: Unknown or unspecified.

LMI_EVENT
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.4 Restoration Service Primitives

The restoration service primitives permit the SLS user to perform functions necessary for BSNT retrieval to initiate or respond to sequenced changeover, buffer updating to respond to sequenced or time-controlled changeover, and buffer clearing to respond to time-controlled changeover or processor outage related failures.

These service primitives implement the restoration services (see Section 3.2.4 [Restoration Services], page 20).

4.2.4.1 SL_RETRIEVE_BSNT_REQ

Description

The retrieve BSNT request service primitive allows the SLS user to request retrieval of the BSNT (backward sequence number transmitted) which indicates the sequence number of the remove message signal unit sent that was last acknowledged. This function is necessary to properly generate or respond to a sequenced changeover procedure by the SLS user.

Format

The retrieve BSNT request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_retrieve_bsnt_req_t;
```

Parameters

The retrieve BSNT request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_RETRIEVE_BSNT_REQ`.

State

This primitive is valid only in the `LMI_ENABLED` management state. The primitive is valid in the `SL_STATE_OUT_OF_SERVICE` state.

New State

The new state is unchanged.

Rules

The SLS user should observe the following rules when issuing the retrieve BSNT request service primitive:

— The SLS user should ensure that the link is in the `SL_STATE_OUT_OF_SERVICE` state before issuing this primitive. One easy way to ensure that the link is in this state is to issue the stop request `SL_STOP_REQ`.

Response

This service primitive requires the SLS provider to acknowledge success of failure of the retrieval operation.

- **Successful retrieval:** When successful, the SLS provider indicate the retrieved BSNT value using the `SL_BSNT_IND` primitive containing the BSNT value. The management and link states remain the same.
- **Unsuccessful retrieval:** When unsuccessful, the SLS provider indicates that the BSNT value cannot be retrieved using the `SL_BSNT_NOT_RETRIEVABLE_IND`. The management and link states remain the same.
- **Non-fatal errors:** When a non-fatal error occurs, the SLS provider indicates the error using the `LMI_ERROR_ACK` primitive containing the error and the reason.

When the management state is `LMI_ENABLED` and the link state is other than `SL_STATE_OUT_OF_SERVICE`, the SLS provider should respond with `SL_BSNT_NOT_RETRIEVABLE_IND` instead of generating a non-fatal error.

Reason for Failure

Most SLS providers are always successful in retrieving the BSNT value. Applicable reasons for failing to retrieve the BSNT value are as follows:

1. Hardware failure.
2. The signalling link is in the incorrect state (e.g. the in-service state).

Applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.4.2 SL_BSNT_IND

Description

The BSNT indication service primitive is originated by the SLS provider to indicate the retrieved BSNT value in response to a SL_RETRIEVE_BSNT_REQ primitive from the SLS user.

Format

The BSNT indication service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_bsnt;
} sl_bsnt_ind_t;
```

Parameters

The BSNT indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_BSNT_IND.

sl_bsnt
: Indicates the value of the BSNT. The format of the BSNT value is provider-specific but is typically a 7-bit or 12-bit sequence number.

State

This primitive is valid in management state LMI_ENABLED and link state SL_STATE_OUT_OF_SERVICE.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing a BSNT indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

— The primitive is only issued in response to an outstanding SL_RETRIEVE_BSNT_REQ primitive when it is possible for the SLS provider to retrieve the BSNT value.

Response

The primitive does not require a response from the SLS user.

4.2.4.3 SL_BSNT_NOT_RETRIEVABLE_IND

Description

The BSNT not retrievable indication service primitive is originated by the SLS provider to indicate that the BSNT value cannot be retrieved in response to a SL_RETRIEVE_BSNT_REQ primitive from the SLS user.

Format

The BSNT not retrievable indication service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_bsnt;
} sl_bsnt_not_retr_ind_t;
```

Parameters

The BSNT not retrievable indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_BSNT_NOT_RETRIEVABLE_IND.

sl_bsnt
: Indicates the value of the BSNT. This value is the known value of the last acknowledged message signal unit from the remote peer or minus one (-1UL) indicating that a reasonable BSNT value is not known. The format of the BSNT is provider-specific, but is typically a 7-bit or 12-bit sequence number.

State

This primitive is valid in management state LMI_ENABLED and is valid in any link state.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing the BSNT not retrievable indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state, but may be issued from any link state.

— The primitive is only issued in response to an outstanding SL_RETRIEVE_BSNT_REQ primitive when it is not possible for the SLS provider to retrieve the BSNT value.

— When issued, a non-fatal error for the same request will not be issued.

Response

The primitive does not require a response from the SLS user.

4.2.4.4 SL_RETRIEVAL_REQUEST_AND_FSNC_REQ

Description

The retrieval request and FSNC request service primitive is originated by the SLS user when it wishes to update the retransmission buffer with the last known acknowledged message (FSNC). The last known acknowledged message is acquired by the SLS user with the sequence changeover procedure of the message transfer part. The primitive requests that the SLS provider update the retransmission buffer and then deliver the contents of the updated retransmission buffer and transmit buffers to the SLS user.

Format

The retrieval request and FSNC request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_fsnc;
} sl_retrieval_req_and_fsnc_t;
```

Parameters

The retrieval request and FSNC request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_RETRIEVAL_REQ_AND_FSNC_REQ`.

`sl_fsnc`
: Specifies the value of the FSNC (forward sequence number confirmed). This is the last known message to be acknowledge by the remote SLS provider. The format of the FSNC is provider-specific, but is typically a 7-bit or 12-bit sequence number.

State

This primitive is only valid in management state `LMI_ENABLED` and is valid in link state `SL_STATE_OUT_OF_SERVICE`.

New State

The new state remains unchanged.

Rules

Response

The retrieval request and FSNC request service primitive request the SLS provider to acknowledge the result of the retrieval action as follows:

– **Successful retrieval:** When successful, the SLS provider indicates the updated contents of the retransmission buffer and the contents of the transmission buffer using the `SL_RETRIEVED_MESSAGE_IND` primitive followed by a `SL_RETRIEVAL_COMPLETE_IND` primitive. The state remains unchanged.

- **Unsuccessful retrieval:** When unsuccessful, the SLS provider indicates failure to retrieve the contents of the buffers with the `SL_RETRIEVAL_NOT_POSSIBLE_IND` primitive.
- **Non-fatal errors:** When a non-fatal error occurs, the SLS provider indicates the error using the `LMI_ERROR_ACK` primitive containing the error and the reason. The state remains unchanged.

When the management stat is `LMI_ENABLED` and the link state is other than `SL_STATE_OUT_OF_SERVICE`, the SLS provider should respond with `SL_RETRIEVAL_NOT_POSSIBLE_IND` instead of generating a non-fatal error.

Reason for Failure

Most SLS providers are always successful in retrieving the updated contents of the retransmission buffer and transmission buffer. Applicable reasons for failing to retrieve the updated buffer contents are as follows:

1. Hardware failure.
2. The signalling link is in the incorrect link state (e.g. the in-service state).
3. The specified value of FSNC does not match and is not adjacent to a message contained in the retransmission buffer.

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.4.5 SL_RETRIEVED_MESSAGE_IND

Description

The retrieved message indication service primitive is originated by the SLS provider to transfer the contents of the updated retransmission buffer and transmission buffer to the SLS user. One primitive is used for each message retrieved. The oldest message in the buffers is indicated first.

Format

The retrieved message indication service primitive consists of one M_PROTO message block followed by one or more M_DATA message blocks containing the retrieved message signal unit in the same format as it was presented to the SLS provider for transmission. The M_PROTO message block is structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_mp;
} sl_retrieved_msg_ind_t;
```

Parameters

The retrieve message indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_RETRIEVED_MESSAGE_IND.

sl_mp
: Indicates the message priority for the message that was specified in the SL_PDU_REQ primitive from the SLS user when the message was submitted for transmission. Message priorities are provider-specific, but are typically between 0 and 3. This message priority field is only applicable to SS7 protocol variants that place message priority bits in a field of the Level 2 header, such as TTC.

State

This primitive is only issued in management state LMI_ENABLED and link state SL_STATE_OUT_OF_SERVICE.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing a retrieved message indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

— The primitive is only issued in response to an outstanding SL_RETRIEVAL_REQUEST_AND_FSNC_REQ primitive when it is possible for the SLS provider to update and retrieve message signal units from the retransmission and transmission buffers.

— The primitive is not issued when the updated retransmission buffer and transmission buffer are empty.

Response

This primitive does not require response from the SLS user.

4.2.4.6 SL_RETRIEVAL_COMPLETE_IND

Description

The retrieval complete indication service primitive is originated by the SLS provider to indicate the completion of transfer of the contents of the updated retransmission buffer and transmission buffer to the SLS user. The primitive is issued in response to a SL_RETRIEVAL_REQUEST_AND_FSNC_REQ primitive issued by the SLS user.

Format

The retrieval complete indication service primitive consists of one M_PROTO message block and zero or more M_DATA message blocks containing the last retrieved message signal unit in the same format as it was presented to the SLS provider for transmission. The M_PROTO message block is structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_mp;
} sl_retrieval_comp_ind_t;
```

Parameters

The retrieval complete indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_RETRIEVAL_COMPLETE_IND.

sl_mp
: When accompanied by M_DATA message blocks containing the last retrieved message signal unit, the sl_mp field indicates the message priority for that message that was specified in the SL_PDU_REQ primitive from the SLSL user when the message was submitted for transmission. Message priorities are provider-specific, but are typically between 0 and 3. This message priority field is only applicable to SS7 protocol variants that place message priority bits in a field of the Level 2 header, such as TTC.

State

This primitive is only issued in management state LMI_ENABLED and link state SL_STATE_OUT_OF_SERVICE.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing a retrieval complete indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

— The primitive is only issued in response to an outstanding `SL_RETRIEVAL_REQUEST_AND_FSNC_REQ` primitive when transfer of the updated retransmission buffer and transmission buffer is complete.

— A message signal unit is not attached to the primitive in `M_DATA` message blocks when the updated retransmission and transmission buffers were empty.

— Attaching the last retrieved message to the primitive in `M_DATA` message blocks is optional and not recommended: the `SL_RETRIEVED_MESSAGE_IND` primitive should be used to transfer all retrieved message signal units first.

— Upon receipt of the retrieval complete indication service primitive, the SLS user will consider the retrieval operation complete.

Response

This primitive does not require a response from the SLS user.

Reason for Failure

4.2.4.7 SL_RETRIEVAL_NOT_POSSIBLE_IND

Description

The retrieval not possible indication service primitive is originated by the SLS provider to indicate that the updated contents of the retransmission and transmission buffers is not possible. The primitive is issued in response to a SL_RETREIVAL_REQUEST_AND_FSNC_REQ primitive received from the SLS user.

Format

The retrieval not possible indication service primitive consists of one M_PROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_retrieval_not_poss_ind_t;
```

Parameters

The retrieval not possible indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_RETRIEVAL_NOT_POSSIBLE_IND.

State

This primitive is only issued from the LMI_ENABLED management state, but may be issued from any link state.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing the retrieval not possible indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state, but may be issued from any link state.

— The primitive is only issued in response to an outstanding SL_RETRIEVAL_REQUEST_AND_FSNC_REQ primitive when it is not possible to update and retrieve the updated contents of the retransmission and transmission buffers.

— When issued, a non-fatal error will not be issued for the same request.

— Upon receipt of the primitive, the SLS user shall consider the retrieval operation complete.

Response

The primitive does not require a response from the SLS user.

4.2.4.8 SL_CLEAR_BUFFERS_REQ

Description

The clear buffers request service primitive is originated by the SLS user to request that all message buffers be cleared by the SLS provider. This includes receive buffer, retransmission buffer and transmission buffers.

Format

The clear buffers request service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_clear_buffers_req_t;
```

Parameters

The clear buffers request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_CLEAR_BUFFERS_REQ.

State

This primitive is only valid in the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

New State

The new state remains unchanged.

Response

The clear buffers request service primitive requires the SLS provider to indicate when the receive buffer and retransmission buffers are cleared, as follows:

- **Successful:** When successful, the SLS provider clears the receive buffer, retransmission buffer and transmission buffer. When the receive buffer is cleared, the SLS provider indicates the clearing with the SL_RB_CLEARED_IND primitive. When the retransmission buffer is cleared, the SLS provider indicates the clearing with the SL_RTB_CLEARED_IND primitive. The state remains unchanged.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using the LMI_ERROR_ACK primitive containing the error and reason for failure. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

LMI_UNSPEC
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.4.9 SL_CLEAR_RTB_REQ

Description

The clear RTB request service primitive is originated by the SLS user to request that only the retransmission buffer be cleared by the SLS provider. This primitive is used in conjunction with the time-controlled changeover procedure of the message transfer part.

Format

The clear RTB request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_clear_rtb_req_t;
```

Parameters

The clear RTB request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_CLEAR_RTB_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state and the `SL_STATE_OUT_OF_SERVICE` link state.

New State

The new state remains unchanged.

Response

The clear RTB request service primitive requires the SLS provider to indicate when the retransmission buffer has been cleared, as follows:

- **Successful:** When successful, the SLS provider clears the retransmission buffer. When the retransmission buffer is cleared, the SLS provider indicates the clearing with the `SL_RTB_CLEARED_IND` primitive. The state remains unchanged.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using the `LMI_ERROR_ACK` primitive containing the error and reason for failure. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.4.10 SL_RB_CLEARED_IND

Description

The RB cleared indication service primitive is originated by the SLS provider whenever the receive buffer has been cleared; either in response to a SL_CLEAR_BUFFERS_REQ primitive from the SLS user, or due to internal state machine operations.

Format

The RB cleared indication service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_rb_cleared_ind_t;
```

Parameters

The RB cleared indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_RB_CLEARED_IND.

State

This primitive is only issued by the SLS provider in the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing the RB cleared indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

— The primitive is issued in response to a SL_CLEAR_BUFFERS_REQ primitive from the SLS user.

— The primitive is also issued in response to internal state machine transitions.

Response

This primitive does not require a response from the SLS user.

4.2.4.11 SL_RTB_CLEARED_IND

Description

The RTB cleared indication service primitive is originated by the SLS provider whenever the retransmission buffer has been cleared; either in response to a SL_CLEAR_BUFFERS_REQ or SL_CLEAR_RTB_REQ primitive, or due to internal state machine operations.

Format

The RTB cleared indication service primitive consists of one M_PROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_rtb_cleared_ind_t;
```

Parameters

The RTB cleared indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_RTB_CLEARED_IND.

State

The primitive is only issued by the SLS provider from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

New State

The new state remains unchanged.

Rules

The SLS provider observes the following rules when issuing the RTB cleared indication service primitive:

— The primitive is only issued from the LMI_ENABLED management state and the SL_STATE_OUT_OF_SERVICE link state.

— The primitive is issued in response to a SL_CLEAR_BUFFERS_REQ or SL_CLEAR_RTB_REQ primitive from the SLS user.

— The primitive is also issued in response to internal state machine transitions.

Response

This primitive does not require a response from the SLS user.

4.2.5 Processor Outage Service Primitives

The processor outage service primitive permit the SLS user the ability to assert and resume from a local processor outage condition as well as being informed by the SLS provider when a local or remote processor outage condition is in effect or has cleared. The SLS user is also able, using these and other primitives, to recover from a local or remote processor outage condition.

These service primitives implement the processor outage services (see Section 3.2.5 [Processor Outage Services], page 23).

4.2.5.1 SL_LOCAL_PROCESSOR_OUTAGE_REQ

Description

The local processor outage request service primitive allows the SLS user to specify that a local processor outage condition exists.

Format

The local processor outage request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_local_proc_outage_req_t;
```

Parameters

The local processor outage request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_LOCAL_PROCESSOR_OUTAGE_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state but is valid from any link state.

New State

The new state is `SL_STATE_PROCESSOR_OUTAGE`.

Response

This primitive does not request a response from the SLS provider.

- **Successful:** When successful, the link moves to the `SL_STATE_PROCESSOR_OUTAGE` state and a local processor outage condition is asserted.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider will negatively acknowledge the primitive using the `LMI_ERROR_ACK` primitive containing the error and reason for failure. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.5.2 SL_LOCAL_PROCESSOR_OUTAGE_IND

Description

The local processor outage indication service primitive is originated by the SLS provider when it detects a local processor outage condition internal to the SLS provider.

Format

The local processor outage indication service primitive consists of on `M_PROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
} sl_loc_proc_out_ind_t;
```

Parameters

The local processor outage indication service primitive contains the following parameters:

`sl_primitive`
: Indicates the service primitive type. Always `SL_LOCAL_PROCESSOR_OUTAGE_IND`.

`sl_timestamp`
: Indicates the time at which the detection of local processor outage occurred. This is UNIX time from epoch timestamp in milliseconds.

State

This primitive is only issued by the SLS provider in the `LMI_ENABLED` management state and active or blocked link state.

New State

The new state is `SL_STATE_PROCESSOR_OUTAGE`.

Rules

The SLS provider observes the following rules when issuing the local processor outage indication service primitive:

— The primitive is only issued in the `LMI_ENABLED` management state.

— SLS provider detection of local processor outage and SLS user detection of local processor outage are independent conditions.

— The SLS provider will issue a `SL_LOCAL_PROCESSOR_RECOVERED_IND` primitive when the local processor outage condition is no longer in effect.

Response

This primitive does not require a response from the SLS user.

4.2.5.3 SL_RESUME_REQ

Description

The resume request service primitive allows the SLS user to specify that a local processor outage condition is no longer in effect. That is, that the local processor has recovered.

Format

The resume request service primitive consists of one M_PROTO or M_PCPROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_resume_req_t;
```

Parameters

The resume request service primitive contains the following parameters:

sl_primitive
: Specifies the service primitive type. Always SL_RESUME_REQ.

State

This primitive is only valid in the LMI_ENABLED management state and when the link is in the SL_STATE_PROCESSOR_OUTAGE state with a local processor outage condition asserted by the SLS user with a previous SL_LOCAL_PROCESSOR_OUTAGE_REQ primitive.

New State

The new state is SL_STATE_IN_SERVICE provided that no other processor outage condition is currently asserted.

Response

This primitive does not request a response from the SLS provider.

- **Successful:** When successful, the link moves to the SL_STATE_IN_SERVICE state and the local processor outage condition is removed.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider will negatively acknowledge the primitive using the LMI_ERROR_ACK primitive containing the error and reason for failure. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

LMI_UNSPEC
: Unknown or unspecified.

LMI_DISC Disconnected.

LMI_EVENT
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.5.4 SL_LOCAL_PROCESSOR_RECOVERED_IND

Description

The local processor recovered indication service primitive is originated by the SLS provider when it detects a remote processor recovery condition.

Format

The local processor recovered indication service primitive consists of one M_PROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
} sl_loc_proc_recovered_ind_t;
```

Parameters

The local processor recovered indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_LOCAL_PROCESSOR_RECOVERED_IND.

sl_timestamp
: Indicates the time at which the detection of local processor recovery occurred. This is UNIX time from epoch timestamp in milliseconds.

State

This primitive is only issued by the SLS provider in the LMI_ENABLED management state and the link state of SL_STATE_PROCESSOR_OUTAGE with local outage asserted by the SLS provider.

New State

The new state is SL_STATE_IN_SERVICE provided that no other processor outage condition (SLS user local, or remote) exists.

Rules

The SLS provider observes the following rules when issuing a local processor recovered indication service primitive:

— The primitive is only issued in the LMI_ENABLED management state.

— The SLS provider will only issue this primitive after it has issued a SL_LOCAL_PROCESSOR_OUTAGE_IND primitive and when the local processor outage condition is no longer in effect.

Response

This primitive does not require a respnose from the SLS user, nevertheless, the SLS user will typically attempt to continue on the link or restore it using restoration service primitives.

4.2.5.5 SL_REMOTE_PROCESSOR_OUTAGE_IND

Description

The remote processour outage indication service primitive is originated by the SLS provider when it detects a remote processor outage condition.

Format

The remove processor outage indication service primitive consists of one `M_PROTO` message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
} sl_rem_proc_out_ind_t;
```

Parameters

The remove processor outage indication service primitive contains the following parameters:

`sl_primitive`
: Indicates the service primitive type. Always `SL_REMOTE_PROCESSOR_OUTAGE_IND`.

`sl_timestamp`
: Indicates the time at which the detection of remote processor outage occurred. This is UNIX time from epoch timestamp in milliseconds.

State

This primitive is only issued by the SLS provider in the `LMI_ENABLED` management state and active or blocked link state.

New State

The new state is `SL_STATE_PROCESSOR_OUTAGE`.

Rules

The SLS provider observes the following rules when issuing the remote processor outage indication service primitive:

— The primitive is only issued in the `LMI_ENABLED` management state.

— The SLS provider will issue a `SL_REMOTE_PROCESSOR_RECOVERED_IND` primitive when the remote processor outage condition is no longer in effect.

Response

This primitive does not require a response from the SLS user.

4.2.5.6 SL_REMOTE_PROCESSOR_RECOVERED_IND

Description

The remote processor recovered indication service primitive is originated by the SLS provider when it detects a remote processor recovery condition.

Format

The remote processor recovered indication service primitive consists of one M_PROTO message block, structured as follows:

```
typedef struct {
    sl_long sl_primitive;
    sl_ulong sl_timestamp;
} sl_rem_proc_recovered_ind_t;
```

Parameters

The remote processor recovered indication service primitive contains the following parameters:

sl_primitive
: Indicates the service primitive type. Always SL_REMOTE_PROCESSOR_RECOVERED_IND.

sl_timestamp
: Indicates the time at which the detection of remote processor recovery occurred. This is UNIX time from epoch timestamp in milliseconds.

State

This primitive is only issued by the SLS provider in the LMI_ENABLED management state and the link state of SL_STATE_PROCESSOR_OUTAGE with remote process outage asserted.

New State

The new state is unchanged.

Rules

The SLS provider observes the following rules when issuing a remote processor recovered indication service primitive:

— The primitive is only issued in the LMI_ENABLED management state.

— The SLS provider will only issue this primitive after it was issued a SL_REMOTE_PROCESSOR_OUTAGE_IND primitive when the remote processor outage condition is no longer in effect.

Response

This primitive does not require a response from the SLS user, nevertheless, the SLS user will typically attempt to continue on the link or restore it using restoration service primitives.

4.2.5.7 SL_CONTINUE_REQ

Description

The continue request service primitive is originated by the SLS user to request that a link previously in a remote processor outage condition, or a SLS provider detected local process outage condition, be continued. This action is normally performed where processor outage has not been of a long duration and it is not necessary to fail or otherwise restore the signalling link.

Format

The continue request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, formatted as follows:

```
typedef struct {
    sl_long sl_primitive;
} sl_continue_req_t;
```

Parameters

The continue request service primitive contains the following parameters:

`sl_primitive`
: Specifies the service primitive type. Always `SL_CONTINUE_REQ`.

State

This primitive is only valid in the `LMI_ENABLED` management state and valid in the `SL_STATE_PROCESSOR_OUTAGE` state where local (SLS provider detected) or remote processor recovery has been indicated.

New State

The new state is `SL_STATE_IN_SERVICE`, provided that there is no other processor outage condition in effect.

Response

This primitive does not require receipt acknowledgement by the SLS provider.

- **Successful:** When successful, the primtiive does not require acknowledgement and the link moves to the `SL_STATE_IN_SERVICE` state.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider negatively acknowledges the primitive using an `LMI_ERROR_ACK` primitive containing the error and reason for failure. The state reamins unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.6 Link Option Management Service Primitives

The link option management service primitives provide another mechanism for options management separate from the local management interface (i.e. the `LMI_OPTMGMT_REQ` and `LMI_OPTMGMT_ACK` primitives). These service primitives are not currently supported by any SLS provider and their use is *deprecated*.

These service primitives implement the link option management service (see Section 3.2.6 [Link Option Management Service], page 25).

4.2.6.1 SL_OPTMGMT_REQ

Description

This SLS user originated primitive requests that the SLS provider options be managed.

Format

The link option management request service primitive consists of one `M_PROTO` or `M_PCPROTO` message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_opt_length;
    lmi_ulong lmi_opt_offset;
    lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_req_t;
```

Parameters

The link option management request service primitive contains the following parameters:

`lmi_primitive`
: Specifies the service primitive type. Always `SL_OPTMGMT_REQ`.

`lmi_opt_length`
: Specifies the length of the options.

`lmi_opt_offset`
: Specifies the offset, from the beginning of the `M_PROTO` message block, of the start of the options.

`lmi_mgmt_flags`
: Specifies the management flags which determine what operation the LMS provider is expected to perform on the specified options. This field can assume one of the following values:

  `LMI_NEGOTIATE`
  : Negotiate the specified value of each specified option and return the negotiated value.

`LMI_CHECK`
: Check the validity of the specified value of each specified option and return the result. Do not alter the current value assumed by the LMS provider.

`LMI_DEFAULT`
: Return the default value for the specified options (or all options). Do not alter the current value assumed by the LMS provider.

`LMI_CURRENT`
: Return the current value for the specified options (or all options). Do not alter the current value assumed by the LMS provider.

State

This primitive is valid in any state where a local acknowledgement is not pending.

New State

The new state remains unchanged.

Rules

Response

The link option management request service primitive requires the LMS provider to acknowledge receipt of the primitive as follows:

- **Successful**: Upon success, the LMS provider acknowledges receipt of the service primitive and successful completion of the link options management service with an `SL_OPTMGMT_ACK` primitive containing the link options management result. The state remains unchanged.
- **Unsuccessful (non-fatal errors)**: Upon failure, the LMS provider acknowledges receipt of the service primitive and failure to complete the link options management service with an `LMI_ERROR_ACK` primitive containing the error. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_BADADDRESS`
: Address was invalid.

`LMI_BADADDRTYPE`
: Invalid address type.

`LMI_BADDIAL`
: (Not used.)

`LMI_BADDIALTYPE`
: (Not used.)

`LMI_BADDISPOSAL`
: Invalid disposal parameter.

`LMI_BADFRAME`
: Defective SDU received.

`LMI_BADPPA`
: Invalid PPA identifier.

`LMI_BADPRIM`
: Unrecognized primitive.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_INITFAILED`
: Link initialization failed.

`LMI_NOTSUPP`
: Primitive not supported by this device.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_WRITEFAIL`
: Unitdata request failed.

`LMI_CRCERR`
: CRC or FCS error.

`LMI_DLE_EOT`
: DLE EOT detected.

`LMI_FORMAT`
: Format error detected.

`LMI_HDLC_ABORT`
: Aborted frame detected.

`LMI_OVERRUN`
: Input overrun.

`LMI_TOOSHORT`
: Frame too short.

`LMI_INCOMPLETE`
: Partial frame received.

`LMI_BUSY` Telephone was busy.

`LMI_NOANSWER`
: Connection went unanswered.

`LMI_CALLREJECT`
: Connection rejected.

`LMI_HDLC_IDLE`
: HDLC line went idle.

`LMI_HDLC_NOTIDLE`
: HDLC link no longer idle.

`LMI_QUIESCENT`
: Line being reassigned.

`LMI_RESUMED`
: Line has been reassigned.

`LMI_DSRTIMEOUT`
: Did not see DSR in time.

`LMI_LAN_COLLISIONS`
: LAN excessive collisions.

`LMI_LAN_REFUSED`
: LAN message refused.

`LMI_LAN_NOSTATION`
: LAN no such station.

`LMI_LOSTCTS`
: Lost Clear to Send signal.

`LMI_DEVERR`
: Start of device-specific error codes.

4.2.6.2 SL_OPTMGMT_ACK

Description

This LMS provider originated primitive is issued by the LMS provider upon successful completion of the link options management service. It indicates the outcome of the link options management operation requested by the LMS user in a SL_OPTMGMT_REQ primitive.

Format

The link option management acknowledgement service primitive consists of one M_PCPROTO message block, structured as follows:

```
typedef struct {
    lmi_long lmi_primitive;
    lmi_ulong lmi_opt_length;
    lmi_ulong lmi_opt_offset;
    lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_ack_t;
```

Parameters

The link option management acknowledgement service primitive contains the following parameters:

lmi_primitive
: Indicates the service primitive type. Always SL_OPTMGMT_ACK.

lmi_opt_length
: Indicates the length of the returned options.

lmi_opt_offset
: Indicates the offset of the returned options from the start of the M_PCPROTO message block.

lmi_mgmt_flags
: Indicates the returned management flags. These flags indicate the overall success of the link options management service. This field can assume one of the following values:

  LMI_SUCCESS
  : The LMS provider succeeded in negotiating or returning all of the options specified by the LMS user in the LMI_OPTMGMT_REQ primitive.

  LMI_FAILURE
  : The LMS provider failed to negotiate one or more of the options specified by the LMS user.

  LMI_PARTSUCCESS
  : The LMS provider negotiated a value of lower quality for one or more of the options specified by the LMS user.

`LMI_READONLY`
: The LMS provider failed to negotiate one ore more of the options specified by the LMS user because the option is treated as read-only by the LMS provider.

`LMI_NOTSUPPORT`
: The LMS provider failed to recognize one or more of the options specified by the LMS user.

State

This primitive is issued by the LMS provider in direct response to an `SL_OPTMGMT_REQ` primitive.

New State

The new state remains unchanged.

Rules

The LMS provider follows the following rules when processing link option management service requests:

— When the `lmi_mgmt_flags` field in the `SL_OPTMGMT_REQ` primitive is set to `LMI_NEGOTIATE`, the LMS provider will attempt to negotiate a value for each of the options specified in the request.

— When the flags are `LMI_DEFAULT`, the LMS provider will return the default values of the specified options, or the default values of all options known to the LMS provider if no options were specified.

— When the flags are `LMI_CURRENT`, the LMS provider will return the current values of the specified options, or all options.

— When the flags are `LMI_CHECK`, the LMS provider will attempt to negotiate a value for each of the options specified in the request and return the resulg of the negotiation, but will not affect the current value of the option.

4.2.7 Event Notification Service Primitives

The event notification service primitives provide another mechanism for event notification separate from the local management interface (i.e. the `LMI_EVENT_IND` primitive). These service primitives are not currently supported by any SLS provider and their use is *deprecated*.

These service primitives implement the event notification service (see Section 3.2.7 [Event Notification Service], page 26).

4.2.7.1 SL_NOTIFY_REQ

Description

This SLS user originated primitives requests that the SLS provider register the SLS user for various events.

Format

Not documented.

Parameters

`sl_primitive`
: Specifies the service primitive type. Always `SL_NOTIFY_REQ`.

State

Any state.

New State

Unchanged.

Response

This primitive does not require receipt acknolwedgement from the SLS provider.

- **Successful:** When successful, the events are registered and no acknowledgement is required. The state remains unchanged.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the SLS provider generates a negative acknowledgement using a `LMI_ERROR_ACK` primitive containing the error and reason for failure. The state remains unchanged.

Reason for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

`LMI_UNSPEC`
: Unknown or unspecified.

`LMI_DISC` Disconnected.

`LMI_EVENT`
: Protocol-specific event occurred.

`LMI_FATALERR`
: Device has become unusable.

`LMI_OUTSTATE`
: Primitive was issued from invalid state.

`LMI_PROTOSHORT`
: `M_PROTO` block too short.

`LMI_SYSERR`
: UNIX system error.

`LMI_DEVERR`
: Start of device-specific error codes.

Notes

This primitive is *deprecated* and has been replaced by the local management inteface event reporting service discussed in Section 3.1.8 [Event Reporting Service], page 15.

4.2.7.2 SL_NOTIFY_IND

Description

This SLS provider originated primitive indicates that an event for which the SLS provider has registered has occurred.

Format

Not documented.

Parameters

`sl_primitive`
: Specifies the service primitive type. Always `SL_NOTIFY_IND`.

State

Any state.

New State

Unchanged.

Rules

The SLS provider observes the following rules when issuing the event notification indication service primtiive:

— This primitive is only issued by the SLS provider for event for which the SLS user has explicitly registered with the `SL_NOTIFY_REQ` primitive.

— Specific events are provider-specific.

Notes

This primitive is *deprecated* and has been replaced by the local management inteface event reporting service discussed in Section 3.1.8 [Event Reporting Service], page 15.

5 Diagnostics Requirements

Two error handling facilities should be provided to the SLS user: one to handle non-fatal errors, and the other to handle fatal errors.

5.1 Non-Fatal Error Handling Facility

These are errors that do not change the state of the SLS interface as seen by the SLS user and provide the user with the option of reissuing the SL primitive with the corrected options specification. The non-fatal error handling is provided only to those primitives that require acknowledgements, and uses the `LMI_ERROR_ACK` to report these errors. These errors retain the state of the SLS interface the same as it was before the SL provider received the primitive that was in error. Syntax errors and rule violations are reported via the non-fatal error handling facility.

5.2 Fatal Error Handling Facility

These errors are issued by the SL provider when it detects errors that are not correctable by the SL user, or if it is unable to report a correctible error to the SLS user. Fatal errors are indicated via the STREAMS message type `M_ERROR` with the UNIX system error `EPROTO`. The `M_ERROR` STREAMS message type will result in the failure of all the UNIX system calls on the stream. The SLS user can recover from a fatal error by having all the processes close the files associated with the stream, and then reopening them for processing.

Appendix A LMI Header File Listing

```
/*****************************************************************************

 @(#) lmi.h,v 0.9.2.1 2007/08/13 19:55:44 brian Exp

 -----------------------------------------------------------------------------

 Copyright (c) 2001-2007  OpenSS7 Corporation <http://www.openss7.com/>
 Copyright (c) 1997-2001  Brian F. G. Bidulock <bidulock@openss7.org>

 All Rights Reserved.

 This program is free software; you can redistribute it and/or modify it under
 the terms of the GNU General Public License as published by the Free Software
 Foundation; version 3 of the License.

 This program is distributed in the hope that it will be useful, but WITHOUT
 ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
 details.

 You should have received a copy of the GNU General Public License along with
 this program.  If not, see <http://www.gnu.org/licenses/>, or write to the
 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

 -----------------------------------------------------------------------------

 U.S. GOVERNMENT RESTRICTED RIGHTS.  If you are licensing this Software on
 behalf of the U.S. Government ("Government"), the following provisions apply
 to you.  If the Software is supplied by the Department of Defense ("DoD"), it
 is classified as "Commercial Computer Software" under paragraph 252.227-7014
 of the DoD Supplement to the Federal Acquisition Regulations ("DFARS") (or any
 successor regulations) and the Government is acquiring only the license rights
 granted herein (the license rights customarily provided to non-Government
 users).  If the Software is supplied to any unit or agency of the Government
 other than DoD, it is classified as "Restricted Computer Software" and the
 Government's rights in the Software are defined in paragraph 52.227-19 of the
 Federal Acquisition Regulations ("FAR") (or any successor regulations) or, in
 the cases of NASA, in paragraph 18.52.227-86 of the NASA Supplement to the FAR
 (or any successor regulations).

 -----------------------------------------------------------------------------

 Commercial licensing and support of this software is available from OpenSS7
 Corporation at a fee.  See http://www.openss7.com/

 -----------------------------------------------------------------------------

 Last Modified 2007/08/13 19:55:44 by brian

 -----------------------------------------------------------------------------

 lmi.h,v
 Revision 0.9.2.1  2007/08/13 19:55:44  brian
 - added spec headers
```

```
 Revision 0.9.2.9  2007/08/12 16:19:53  brian
 - new PPA handling

 Revision 0.9.2.8  2007/03/25 18:59:12  brian
 - changes to support 2.6.20-1.2307.fc5 kernel

 Revision 0.9.2.7  2007/01/28 01:09:50  brian
 - updated test programs and working up m2ua-as driver

 *****************************************************************************/

#ifndef __LMI_H__
#define __LMI_H__

#ident "@(#) lmi.h,v (0.9.2.1) Copyright (c) 2001-2007 OpenSS7 Corporation."

/* This file can be processed by doxygen(1). */

#define LMI_PROTO_BASE            16L

#define LMI_DSTR_FIRST          ( 1L + LMI_PROTO_BASE )
#define LMI_INFO_REQ            ( 1L + LMI_PROTO_BASE )
#define LMI_ATTACH_REQ          ( 2L + LMI_PROTO_BASE )
#define LMI_DETACH_REQ          ( 3L + LMI_PROTO_BASE )
#define LMI_ENABLE_REQ          ( 4L + LMI_PROTO_BASE )
#define LMI_DISABLE_REQ         ( 5L + LMI_PROTO_BASE )
#define LMI_OPTMGMT_REQ         ( 6L + LMI_PROTO_BASE )
#define LMI_DSTR_LAST           ( 6L + LMI_PROTO_BASE )

#define LMI_USTR_LAST           (-1L - LMI_PROTO_BASE )
#define LMI_INFO_ACK            (-1L - LMI_PROTO_BASE )
#define LMI_OK_ACK              (-2L - LMI_PROTO_BASE )
#define LMI_ERROR_ACK           (-3L - LMI_PROTO_BASE )
#define LMI_ENABLE_CON          (-4L - LMI_PROTO_BASE )
#define LMI_DISABLE_CON         (-5L - LMI_PROTO_BASE )
#define LMI_OPTMGMT_ACK         (-6L - LMI_PROTO_BASE )
#define LMI_ERROR_IND           (-7L - LMI_PROTO_BASE )
#define LMI_STATS_IND           (-8L - LMI_PROTO_BASE )
#define LMI_EVENT_IND           (-9L - LMI_PROTO_BASE )
#define LMI_USTR_FIRST          (-9L - LMI_PROTO_BASE )

#define LMI_UNATTACHED          1L      /* No PPA attached, awating LMI_ATTACH_REQ */
#define LMI_ATTACH_PENDING      2L      /* Waiting for attach */
#define LMI_UNUSABLE            3L      /* Device cannot be used, STREAM in hung state */
#define LMI_DISABLED            4L      /* PPA attached, awaiting LMI_ENABLE_REQ */
#define LMI_ENABLE_PENDING      5L      /* Waiting to send LMI_ENABLE_CON */
#define LMI_ENABLED             6L      /* Ready for use, awaiting primtiive exchange */
#define LMI_DISABLE_PENDING     7L      /* Waiting to send LMI_DISABLE_CON */
#define LMI_DETACH_PENDING      8L      /* Waiting for detach */

/*
 *  LMI_ERROR_ACK and LMI_ERROR_IND reason codes
 */
#define LMI_UNSPEC              0x00000000      /* Unknown or unspecified */
#define LMI_BADADDRESS          0x00010000      /* Address was invalid */
```

```
#define LMI_BADADDRTYPE         0x00020000      /* Invalid address type */
#define LMI_BADDIAL             0x00030000      /* (not used) */
#define LMI_BADDIALTYPE         0x00040000      /* (not used) */
#define LMI_BADDISPOSAL         0x00050000      /* Invalid disposal parameter */
#define LMI_BADFRAME            0x00060000      /* Defective SDU received */
#define LMI_BADPPA              0x00070000      /* Invalid PPA identifier */
#define LMI_BADPRIM             0x00080000      /* Unregognized primitive */
#define LMI_DISC                0x00090000      /* Disconnected */
#define LMI_EVENT               0x000a0000      /* Protocol-specific event ocurred */
#define LMI_FATALERR            0x000b0000      /* Device has become unusable */
#define LMI_INITFAILED          0x000c0000      /* Link initialization failed */
#define LMI_NOTSUPP             0x000d0000      /* Primitive not supported by this device */
#define LMI_OUTSTATE            0x000e0000      /* Primitive was issued from invalid state */
#define LMI_PROTOSHORT          0x000f0000      /* M_PROTO block too short */
#define LMI_SYSERR              0x00100000      /* UNIX system error */
#define LMI_WRITEFAIL           0x00110000      /* Unitdata request failed */
#define LMI_CRCERR              0x00120000      /* CRC or FCS error */
#define LMI_DLE_EOT             0x00130000      /* DLE EOT detected */
#define LMI_FORMAT              0x00140000      /* Format error detected */
#define LMI_HDLC_ABORT          0x00150000      /* Aborted frame detected */
#define LMI_OVERRUN             0x00160000      /* Input overrun */
#define LMI_TOOSHORT            0x00170000      /* Frame too short */
#define LMI_INCOMPLETE          0x00180000      /* Partial frame received */
#define LMI_BUSY                0x00190000      /* Telephone was busy */
#define LMI_NOANSWER            0x001a0000      /* Connection went unanswered */
#define LMI_CALLREJECT          0x001b0000      /* Connection rejected */
#define LMI_HDLC_IDLE           0x001c0000      /* HDLC line went idle */
#define LMI_HDLC_NOTIDLE        0x001d0000      /* HDLC link no longer idle */
#define LMI_QUIESCENT           0x001e0000      /* Line being reassigned */
#define LMI_RESUMED             0x001f0000      /* Line has been reassigned */
#define LMI_DSRTIMEOUT          0x00200000      /* Did not see DSR in time */
#define LMI_LAN_COLLISIONS      0x00210000      /* LAN excessive collisions */
#define LMI_LAN_REFUSED         0x00220000      /* LAN message refused */
#define LMI_LAN_NOSTATION       0x00230000      /* LAN no such station */
#define LMI_LOSTCTS             0x00240000      /* Lost Clear to Send signal */
#define LMI_DEVERR              0x00250000      /* Start of device-specific error codes */

typedef signed int lmi_long;
typedef unsigned int lmi_ulong;
typedef unsigned short lmi_ushort;
typedef unsigned char lmi_uchar;

/*
 *  LOCAL MANAGEMENT PRIMITIVES
 */

/*
   LMI_INFO_REQ, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_INFO_REQ */
} lmi_info_req_t;

/*
   LMI_INFO_ACK, M_PROTO or M_PCPROTO
```

```
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_INFO_ACK */
        lmi_ulong lmi_version;
        lmi_ulong lmi_state;
        lmi_ulong lmi_max_sdu;
        lmi_ulong lmi_min_sdu;
        lmi_ulong lmi_header_len;
        lmi_ulong lmi_ppa_style;
        lmi_ulong lmi_ppa_length;
        lmi_ulong lmi_ppa_offset;
        lmi_ulong lmi_prov_flags;       /* provider specific flags */
        lmi_ulong lmi_prov_state;       /* provider specific state */
        lmi_uchar lmi_ppa_addr[0];
} lmi_info_ack_t;

#define LMI_VERSION_1       1
#define LMI_VERSION_2       2
#define LMI_CURRENT_VERSION LMI_VERSION_2

/*
 *  LMI provider style.
 *
 *  The LMI provider style which determines whether a provider requires an
 *  LMI_ATTACH_REQ to inform the provider which PPA user messages should be
 *  sent/received on.
 */
#define LMI_STYLE1      0x00    /* PPA is implicitly bound by open(2) */
#define LMI_STYLE2      0x01    /* PPA must be explicitly bound via STD_ATTACH_REQ */

/*
   LMI_ATTACH_REQ, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_ATTACH_REQ */
        lmi_ulong lmi_ppa_length;
        lmi_ulong lmi_ppa_offset;
        lmi_uchar lmi_ppa[0];
} lmi_attach_req_t;

/*
   LMI_DETACH_REQ, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_DETACH_REQ */
} lmi_detach_req_t;

/*
   LMI_ENABLE_REQ, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_ENABLE_REQ */
```

```
        lmi_ulong lmi_rem_length;
        lmi_ulong lmi_rem_offset;
        lmi_uchar lmi_rem[0];
} lmi_enable_req_t;

/*
   LMI_DISABLE_REQ, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_DISABLE_REQ */
} lmi_disable_req_t;

/*
   LMI_OK_ACK, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_OK_ACK */
        lmi_long lmi_correct_primitive;
        lmi_ulong lmi_state;
} lmi_ok_ack_t;

/*
   LMI_ERROR_ACK, M_CTL
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_ERROR_ACK */
        lmi_ulong lmi_errno;
        lmi_ulong lmi_reason;
        lmi_long lmi_error_primitive;
        lmi_ulong lmi_state;
} lmi_error_ack_t;

/*
   LMI_ENABLE_CON, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_ENABLE_CON */
        lmi_ulong lmi_state;
} lmi_enable_con_t;

/*
   LMI_DISABLE_CON, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_DISABLE_CON */
        lmi_ulong lmi_state;
} lmi_disable_con_t;

/*
   LMI_OPTMGMT_REQ, M_PCPROTO
 */
```

```
typedef struct {
        lmi_long lmi_primitive;         /* LMI_OPTMGMT_REQ */
        lmi_ulong lmi_opt_length;
        lmi_ulong lmi_opt_offset;
        lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_req_t;

/*
   LMI_OPTMGMT_ACK, M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_OPMGMT_ACK */
        lmi_ulong lmi_opt_length;
        lmi_ulong lmi_opt_offset;
        lmi_ulong lmi_mgmt_flags;
} lmi_optmgmt_ack_t;

#undef LMI_DEFAULT

#define LMI_NEGOTIATE           0x0004
#define LMI_CHECK               0x0008
#define LMI_DEFAULT             0x0010
#define LMI_SUCCESS             0x0020
#define LMI_FAILURE             0x0040
#define LMI_CURRENT             0x0080
#define LMI_PARTSUCCESS         0x0100
#define LMI_READONLY            0x0200
#define LMI_NOTSUPPORT          0x0400

/*
   LMI_ERROR_IND, M_PROTO or M_PCPROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_ERROR_IND */
        lmi_ulong lmi_errno;
        lmi_ulong lmi_reason;
        lmi_ulong lmi_state;
} lmi_error_ind_t;

/*
   LMI_STATS_IND, M_PROTO
 */

typedef struct {
        lmi_long lmi_primitive;         /* LMI_STATS_IND */
        lmi_ulong lmi_interval;
        lmi_ulong lmi_timestamp;
} lmi_stats_ind_t;

/*
   LMI_EVENT_IND, M_PROTO
 */
```

```
typedef struct {
        lmi_long lmi_primitive;         /* LMI_EVENT_IND */
        lmi_ulong lmi_objectid;
        lmi_ulong lmi_timestamp;
        lmi_ulong lmi_severity;
} lmi_event_ind_t;

union LMI_primitive {
        lmi_long lmi_primitive;
        lmi_ok_ack_t ok_ack;
        lmi_error_ack_t error_ack;
        lmi_error_ind_t error_ind;
        lmi_stats_ind_t stats_ind;
        lmi_event_ind_t event_ind;
};

union LMI_primitives {
        lmi_long lmi_primitive;
        lmi_info_req_t info_req;
        lmi_info_ack_t info_ack;
        lmi_attach_req_t attach_req;
        lmi_detach_req_t detach_req;
        lmi_enable_req_t enable_req;
        lmi_disable_req_t disable_req;
        lmi_ok_ack_t ok_ack;
        lmi_error_ack_t error_ack;
        lmi_enable_con_t enable_con;
        lmi_disable_con_t disable_con;
        lmi_error_ind_t error_ind;
        lmi_stats_ind_t stats_ind;
        lmi_event_ind_t event_ind;
};

#define LMI_INFO_REQ_SIZE       sizeof(lmi_info_req_t)
#define LMI_INFO_ACK_SIZE       sizeof(lmi_info_ack_t)
#define LMI_ATTACH_REQ_SIZE     sizeof(lmi_attach_req_t)
#define LMI_DETACH_REQ_SIZE     sizeof(lmi_detach_req_t)
#define LMI_ENABLE_REQ_SIZE     sizeof(lmi_enable_req_t)
#define LMI_DISABLE_REQ_SIZE    sizeof(lmi_disable_req_t)
#define LMI_OK_ACK_SIZE         sizeof(lmi_ok_ack_t)
#define LMI_ERROR_ACK_SIZE      sizeof(lmi_error_ack_t)
#define LMI_ENABLE_CON_SIZE     sizeof(lmi_enable_con_t)
#define LMI_DISABLE_CON_SIZE    sizeof(lmi_disable_con_t)
#define LMI_ERROR_IND_SIZE      sizeof(lmi_error_ind_t)
#define LMI_STATS_IND_SIZE      sizeof(lmi_stats_ind_t)
#define LMI_EVENT_IND_SIZE      sizeof(lmi_event_ind_t)

typedef struct lmi_opthdr {
        lmi_ulong level;
        lmi_ulong name;
        lmi_ulong length;
        lmi_ulong status;
        lmi_uchar value[0];
        /*
           followed by option value
         */
```

```
} lmi_opthdr_t;

#define LMI_LEVEL_COMMON        '\0'
#define LMI_LEVEL_SDL           'd'
#define LMI_LEVEL_SDT           't'
#define LMI_LEVEL_SL            'l'
#define LMI_LEVEL_SLS           's'
#define LMI_LEVEL_MTP           'M'
#define LMI_LEVEL_SCCP          'S'
#define LMI_LEVEL_ISUP          'I'
#define LMI_LEVEL_TCAP          'T'

#define LMI_OPT_PROTOCOL        1       /* use struct lmi_option */
#define LMI_OPT_STATISTICS      2       /* use struct lmi_sta */

#endif                          /* __LMI_H__ */
```

Appendix B SLI Header File Listing

```
/*****************************************************************************

 @(#) sli.h,v 0.9.2.1 2007/08/13 19:55:44 brian Exp

 -----------------------------------------------------------------------------

 Copyright (c) 2001-2007  OpenSS7 Corporation <http://www.openss7.com/>
 Copyright (c) 1997-2001  Brian F. G. Bidulock <bidulock@openss7.org>

 All Rights Reserved.

 This program is free software; you can redistribute it and/or modify it under
 the terms of the GNU General Public License as published by the Free Software
 Foundation; version 3 of the License.

 This program is distributed in the hope that it will be useful, but WITHOUT
 ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
 details.

 You should have received a copy of the GNU General Public License along with
 this program.  If not, see <http://www.gnu.org/licenses/>, or write to the
 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

 -----------------------------------------------------------------------------

 U.S. GOVERNMENT RESTRICTED RIGHTS.  If you are licensing this Software on
 behalf of the U.S. Government ("Government"), the following provisions apply
 to you.  If the Software is supplied by the Department of Defense ("DoD"), it
 is classified as "Commercial Computer Software" under paragraph 252.227-7014
 of the DoD Supplement to the Federal Acquisition Regulations ("DFARS") (or any
 successor regulations) and the Government is acquiring only the license rights
 granted herein (the license rights customarily provided to non-Government
 users).  If the Software is supplied to any unit or agency of the Government
 other than DoD, it is classified as "Restricted Computer Software" and the
 Government's rights in the Software are defined in paragraph 52.227-19 of the
 Federal Acquisition Regulations ("FAR") (or any successor regulations) or, in
 the cases of NASA, in paragraph 18.52.227-86 of the NASA Supplement to the FAR
 (or any successor regulations).

 -----------------------------------------------------------------------------

 Commercial licensing and support of this software is available from OpenSS7
 Corporation at a fee.  See http://www.openss7.com/

 -----------------------------------------------------------------------------

 Last Modified 2007/08/13 19:55:44 by brian

 -----------------------------------------------------------------------------

 sli.h,v
 Revision 0.9.2.1  2007/08/13 19:55:44  brian
 - added spec headers
```

```
 Revision 0.9.2.7  2007/08/12 16:19:53  brian
 - new PPA handling

 Revision 0.9.2.6  2007/08/03 13:35:01  brian
 - manual updates, put ss7 modules in public release

 Revision 0.9.2.5  2007/06/17 01:56:02  brian
 - updates for release, remove any later language

 *****************************************************************************/

#ifndef __SS7_SLI_H__
#define __SS7_SLI_H__

#ident "@(#) sli.h,v (0.9.2.1) Copyright (c) 2001-2007 OpenSS7 Corporation."

/* This file can be processed by doxygen(1). */

typedef lmi_long sl_long;
typedef lmi_ulong sl_ulong;
typedef lmi_ushort sl_ushort;
typedef lmi_uchar sl_uchar;

#define SL_PROTO_BASE                             64

#define SL_DSTR_FIRST                           (  1 + SL_PROTO_BASE)
#define SL_PDU_REQ                              (  1 + SL_PROTO_BASE)
#define SL_EMERGENCY_REQ                        (  2 + SL_PROTO_BASE)
#define SL_EMERGENCY_CEASES_REQ                 (  3 + SL_PROTO_BASE)
#define SL_START_REQ                            (  4 + SL_PROTO_BASE)
#define SL_STOP_REQ                             (  5 + SL_PROTO_BASE)
#define SL_RETRIEVE_BSNT_REQ                    (  6 + SL_PROTO_BASE)
#define SL_RETRIEVAL_REQUEST_AND_FSNC_REQ       (  7 + SL_PROTO_BASE)
#define SL_CLEAR_BUFFERS_REQ                    (  8 + SL_PROTO_BASE)
#define SL_CLEAR_RTB_REQ                        (  9 + SL_PROTO_BASE)
#define SL_CONTINUE_REQ                         ( 10 + SL_PROTO_BASE)
#define SL_LOCAL_PROCESSOR_OUTAGE_REQ           ( 11 + SL_PROTO_BASE)
#define SL_RESUME_REQ                           ( 12 + SL_PROTO_BASE)
#define SL_CONGESTION_DISCARD_REQ               ( 13 + SL_PROTO_BASE)
#define SL_CONGESTION_ACCEPT_REQ                ( 14 + SL_PROTO_BASE)
#define SL_NO_CONGESTION_REQ                    ( 15 + SL_PROTO_BASE)
#define SL_POWER_ON_REQ                         ( 16 + SL_PROTO_BASE)
#define SL_OPTMGMT_REQ                          ( 17 + SL_PROTO_BASE)
#define SL_NOTIFY_REQ                           ( 18 + SL_PROTO_BASE)
#define SL_DSTR_LAST                            ( 18 + SL_PROTO_BASE)

#define SL_USTR_LAST                            ( -1 - SL_PROTO_BASE)
#define SL_PDU_IND                              ( -1 - SL_PROTO_BASE)
#define SL_LINK_CONGESTED_IND                   ( -2 - SL_PROTO_BASE)
#define SL_LINK_CONGESTION_CEASED_IND           ( -3 - SL_PROTO_BASE)
#define SL_RETRIEVED_MESSAGE_IND                ( -4 - SL_PROTO_BASE)
#define SL_RETRIEVAL_COMPLETE_IND               ( -5 - SL_PROTO_BASE)
#define SL_RB_CLEARED_IND                       ( -6 - SL_PROTO_BASE)
#define SL_BSNT_IND                             ( -7 - SL_PROTO_BASE)
#define SL_IN_SERVICE_IND                       ( -8 - SL_PROTO_BASE)
```

```
#define SL_OUT_OF_SERVICE_IND                   ( -9 - SL_PROTO_BASE)
#define SL_REMOTE_PROCESSOR_OUTAGE_IND          (-10 - SL_PROTO_BASE)
#define SL_REMOTE_PROCESSOR_RECOVERED_IND       (-11 - SL_PROTO_BASE)
#define SL_RTB_CLEARED_IND                      (-12 - SL_PROTO_BASE)
#define SL_RETRIEVAL_NOT_POSSIBLE_IND           (-13 - SL_PROTO_BASE)
#define SL_BSNT_NOT_RETRIEVABLE_IND             (-14 - SL_PROTO_BASE)
#define SL_OPTMGMT_ACK                          (-15 - SL_PROTO_BASE)
#define SL_NOTIFY_IND                           (-16 - SL_PROTO_BASE)
#define SL_LOCAL_PROCESSOR_OUTAGE_IND           (-17 - SL_PROTO_BASE)
#define SL_LOCAL_PROCESSOR_RECOVERED_IND        (-18 - SL_PROTO_BASE)
#define SL_USTR_FIRST                           (-18 - SL_PROTO_BASE)

/*
 *  SLI PROVIDER STATE
 */
#define SLS_POWER_OFF                   0
#define SLS_OUT_OF_SERVICE              1
#define SLS_NOT_ALIGNED                 2
#define SLS_INITIAL_ALIGNMENT           3
#define SLS_PROVING                     4
#define SLS_ALIGNED_READY               5
#define SLS_ALIGNED_NOT_READY           6
#define SLS_IN_SERVICE                  7
#define SLS_PROCESSOR_OUTAGE            8

/*
 *  SLI PROVIDER FLAGS
 */
#define SLF_LOC_PROC_OUT        (1<< 0)
#define SLF_REM_PROC_OUT        (1<< 1)
#define SLF_LOC_IN_SERV         (1<< 2)
#define SLF_REM_IN_SERV         (1<< 3)
#define SLF_LOC_BUSY            (1<< 4)
#define SLF_REM_BUSY            (1<< 5)
#define SLF_LOC_EMERG           (1<< 6)
#define SLF_EMERGENCY           SLF_LOC_EMERG
#define SLF_REM_EMERG           (1<< 7)
#define SLF_RECV_MSU            (1<< 8)
#define SLF_SEND_MSU            (1<< 9)
#define SLF_CONG_ACCEPT         (1<<10)
#define SLF_CONG_DISCARD        (1<<11)
#define SLF_RTB_FULL            (1<<12)
#define SLF_L3_CONG_DETECT      (1<<13)
#define SLF_L2_CONG_DETECT      (1<<14)
#define SLF_LINK_CONGESTED      SLF_L2_CONG_DETECT
#define SLF_CONTINUE            (1<<15)
#define SLF_LEVEL_3_IND         SLF_CONTINUE
#define SLF_CLEAR_RTB           (1<<16)
#define SLF_NEED_FLUSH          (1<<17)
#define SLF_WAIT_SYNC           (1<<18)
#define SLF_REM_ALIGN           (1<<19)

/*
 *  SLI PROTOCOL PRIMITIVES
 */
```

```
/*
 *  SL_PDU_REQ, optional M_PROTO type, with M_DATA block(s)
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_mp;
} sl_pdu_req_t;

/*
 *  SL_PDU_IND, optional M_PROTO type, with M_DATA block(s)
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_mp;
} sl_pdu_ind_t;

/*
 *  PROTOCOL CONTROL PRIMITIVES
 */

/*
 *  SL_EMERGENCY_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_emergency_req_t;

/*
 *  SL_EMERGENCY_CEASES_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_emergency_ceases_req_t;

/*
 *  SL_START_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_start_req_t;

/*
 *  SL_STOP_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_stop_req_t;

/*
 *  SL_RETRIEVE_BSNT_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_retrieve_bsnt_req_t;

/*
```

```
 *  SL_RETRIEVAL_REQUEST_AND_FSNC_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_fsnc;
} sl_retrieval_req_and_fsnc_t;

/*
 *  SL_CLEAR_BUFFERS_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_clear_buffers_req_t;

/*
 *  SL_CLEAR_RTB_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_clear_rtb_req_t;

/*
 *  SL_CONTINUE_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_continue_req_t;

/*
 *  SL_LOCAL_PROCESSOR_OUTAGE_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_local_proc_outage_req_t;

/*
 *  SL_RESUME_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_resume_req_t;

/*
 *  SL_CONGESTION_DISCARD_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_cong_discard_req_t;

/*
 *  SL_CONGESTION_ACCEPT_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_cong_accept_req_t;
```

```
/*
 *  SL_NO_CONGESTION_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_no_cong_req_t;

/*
 * SL_POWER_ON_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_power_on_req_t;

/*
 *  SL_LINK_CONGESTED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
        sl_ulong sl_cong_status;        /* congestion status */
        sl_ulong sl_disc_status;        /* discard status */
} sl_link_cong_ind_t;

/*
 *  SL_LINK_CONGESTION_CEASED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
        sl_ulong sl_cong_status;        /* congestion status */
        sl_ulong sl_disc_status;        /* discard status */
} sl_link_cong_ceased_ind_t;

/*
 *  SL_RETRIEVED_MESSAGE_IND, M_PROTO or M_PCPROTO type with M_DATA block(s)
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_mp;
} sl_retrieved_msg_ind_t;

/*
 *  SL_RETRIEVAL_COMPLETE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_mp;
} sl_retrieval_comp_ind_t;

/*
 *  SL_RETRIEVAL_NOT_POSSIBLE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_retrieval_not_poss_ind_t;
```

```
/*
 *  SL_RB_CLEARED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_rb_cleared_ind_t;

/*
 *  SL_BSNT_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_bsnt;
} sl_bsnt_ind_t;

/*
 *  SL_BSNT_NOT_RETRIEVABLE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_bsnt;
} sl_bsnt_not_retr_ind_t;

/*
 *  SL_IN_SERVICE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_in_service_ind_t;

/*
 *  SL_OUT_OF_SERVICE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
        sl_ulong sl_reason;
} sl_out_of_service_ind_t;

/*
 *  These reasons for failure as so that upstream module can
 *  collect statistics per link per ITU-T Q.752 Table 1
 *  requirements.
 */
#define SL_FAIL_UNSPECIFIED             0x0001
#define SL_FAIL_CONG_TIMEOUT            0x0002
#define SL_FAIL_ACK_TIMEOUT             0x0004
#define SL_FAIL_ABNORMAL_BSNR           0x0008
#define SL_FAIL_ABNORMAL_FIBR           0x0010
#define SL_FAIL_SUERM_EIM               0x0020
#define SL_FAIL_ALIGNMENT_NOT_POSSIBLE  0x0040
#define SL_FAIL_RECEIVED_SIO            0x0080
#define SL_FAIL_RECEIVED_SIN            0x0100
#define SL_FAIL_RECEIVED_SIE            0x0200
#define SL_FAIL_RECEIVED_SIOS           0x0400
```

```
#define SL_FAIL_T1_TIMEOUT              0x0800

/*
 *  SL_REMOTE_PROCESSOR_OUTAGE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
} sl_rem_proc_out_ind_t;

/*
 *  SL_REMOTE_PROCESSOR_RECOVERED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
} sl_rem_proc_recovered_ind_t;

/*
 *  SL_RTB_CLEARED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
} sl_rtb_cleared_ind_t;

/*
 *  SL_LOCAL_PROCESSOR_OUTAGE_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
} sl_loc_proc_out_ind_t;

/*
 *  SL_LOCAL_PROCESSOR_RECOVERED_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        sl_long sl_primitive;
        sl_ulong sl_timestamp;
} sl_loc_proc_recovered_ind_t;

/*
 *  Generic single argument type
 */
typedef struct {
        sl_ulong sl_cmd;
        sl_ulong sl_arg;
} sl_cmd_arg_t;

/*
 *  Generic double argument type
 */
typedef struct {
        sl_ulong sl_cmd;
        sl_ulong sl_arg1;
        sl_ulong sl_arg2;
```

```
} sl_cmd_2arg_t;

/*
 *  Generic triple argument type
 */
typedef struct {
        sl_ulong sl_cmd;
        sl_ulong sl_arg1;
        sl_ulong sl_arg2;
        sl_ulong sl_arg3;
} sl_cmd_3arg_t;

union SL_primitives {
        sl_long sl_primitive;
        sl_cmd_arg_t cmd_arg;
        sl_cmd_2arg_t cmd_2arg;
        sl_cmd_3arg_t cmd_3arg;
        sl_pdu_req_t pdu_req;
        sl_pdu_ind_t pdu_ind;
        sl_emergency_req_t emergency_req;
        sl_emergency_ceases_req_t emergency_ceases_req;
        sl_start_req_t start_req;
        sl_stop_req_t stop_req;
        sl_retrieve_bsnt_req_t retrieve_bsnt_req;
        sl_retrieval_req_and_fsnc_t retrieval_req_and_fsnc;
        sl_resume_req_t resume_req;
        sl_continue_req_t continue_req;
        sl_clear_buffers_req_t clear_buffers_req;
        sl_clear_rtb_req_t clear_rtb_req;
        sl_local_proc_outage_req_t local_proc_outage_req;
        sl_cong_discard_req_t cong_discard_req;
        sl_cong_accept_req_t cong_accept_req;
        sl_no_cong_req_t no_cong_req;
        sl_power_on_req_t power_on_req;
        sl_link_cong_ind_t link_cong_ind;
        sl_link_cong_ceased_ind_t link_cong_ceased_ind;
        sl_retrieved_msg_ind_t retrieved_msg_ind;
        sl_retrieval_comp_ind_t retrieval_comp_ind;
        sl_retrieval_not_poss_ind_t retrieval_not_poss_ind;
        sl_rb_cleared_ind_t rb_cleared_ind;
        sl_bsnt_ind_t bsnt_ind;
        sl_bsnt_not_retr_ind_t bsnt_not_retr_ind;
        sl_in_service_ind_t in_service_ind;
        sl_out_of_service_ind_t out_of_service_ind;
        sl_rem_proc_out_ind_t rem_proc_out_ind;
        sl_rem_proc_recovered_ind_t rem_proc_recovered_ind;
        sl_rtb_cleared_ind_t rtb_cleared_ind;
        sl_loc_proc_out_ind_t loc_proc_out_ind;
        sl_loc_proc_recovered_ind_t loc_proc_recovered_ind;
};

typedef union SL_primitives sl_prim_t;

#define SL_CMD_ARG_SIZE                 sizeof(sl_cmd_arg_t)
#define SL_CMD_2ARG_SIZE                sizeof(sl_cmd_2arg_t)
#define SL_CMD_3ARG_SIZE                sizeof(sl_cmd_3arg_t)
```

```
#define SL_PDU_REQ_SIZE                 sizeof(sl_pdu_req_t)
#define SL_PDU_IND_SIZE                 sizeof(sl_pdu_ind_t)
#define SL_EMERGENCY_REQ_SIZE           sizeof(sl_emergency_req_t)
#define SL_EMERGENCY_CEASES_REQ_SIZE    sizeof(sl_emergency_ceases_req_t)
#define SL_START_REQ_SIZE               sizeof(sl_start_req_t)
#define SL_STOP_REQ_SIZE                sizeof(sl_stop_req_t)
#define SL_RETRIEVE_BSNT_REQ_SIZE       sizeof(sl_retrieve_bsnt_req_t)
#define SL_RETRIEVAL_REQ_AND_FSNC_SIZE  sizeof(sl_retrieval_req_and_fsnc_t)
#define SL_RESUME_REQ_SIZE              sizeof(sl_resume_req_t)
#define SL_CONTINUE_REQ_SIZE            sizeof(sl_continue_req_t)
#define SL_CLEAR_BUFFERS_REQ_SIZE       sizeof(sl_clear_buffers_req_t)
#define SL_CLEAR_RTB_REQ_SIZE           sizeof(sl_clear_rtb_req_t)
#define SL_LOCAL_PROC_OUTAGE_REQ_SIZE   sizeof(sl_local_proc_outage_req_t)
#define SL_CONG_DISCARD_REQ_SIZE        sizeof(sl_cong_discard_req_t)
#define SL_CONG_ACCEPT_REQ_SIZE         sizeof(sl_cong_accept_req_t)
#define SL_NO_CONG_REQ_SIZE             sizeof(sl_no_cong_req_t)
#define SL_POWER_ON_REQ_SIZE            sizeof(sl_power_on_req_t)
#define SL_LINK_CONG_IND_SIZE           sizeof(sl_link_cong_ind_t)
#define SL_LINK_CONG_CEASED_IND_SIZE    sizeof(sl_link_cong_ceased_ind_t)
#define SL_RETRIEVED_MSG_IND_SIZE       sizeof(sl_retrieved_msg_ind_t)
#define SL_RETRIEVAL_COMP_IND_SIZE      sizeof(sl_retrieval_comp_ind_t)
#define SL_RETRIEVAL_NOT_POSS_IND_SIZE  sizeof(sl_retrieval_not_poss_ind_t)
#define SL_RB_CLEARED_IND_SIZE          sizeof(sl_rb_cleared_ind_t)
#define SL_BSNT_IND_SIZE                sizeof(sl_bsnt_ind_t)
#define SL_BSNT_NOT_RETR_IND_SIZE       sizeof(sl_bsnt_not_retr_ind_t)
#define SL_IN_SERVICE_IND_SIZE          sizeof(sl_in_service_ind_t)
#define SL_OUT_OF_SERVICE_SIZE          sizeof(sl_out_of_service_ind_t)
#define SL_REM_PROC_OUT_IND_SIZE        sizeof(sl_rem_proc_out_ind_t)
#define SL_REM_PROC_RECOVERED_IND_SIZE  sizeof(sl_rem_proc_recovered_ind_t)
#define SL_RTB_CLEARED_IND_SIZE         sizeof(sl_rtb_cleared_ind_t)
#define SL_LOC_PROC_OUT_IND_SIZE        sizeof(sl_loc_proc_out_ind_t)
#define SL_LOC_PROC_RECOVERED_IND_SIZE  sizeof(sl_loc_proc_recovered_ind_t)

#define SL_OPT_PROTOCOL         LMI_OPT_PROTOCOL
#define SL_OPT_STATISTICS       LMI_OPT_STATISTICS
#define SL_OPT_CONFIG           3       /* use struct sl_config */
#define SL_OPT_STATEM           4       /* use struct sl_statem */
#define SL_OPT_STATS            5       /* use struct sl_stats */

#endif                          /* __SS7_SLI_H__ */
```

License

GNU Free Documentation License

GNU FREE DOCUMENTATION LICENSE
Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Terms and Conditions for Copying, Distribution and Modification

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a

textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be

added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

END OF TERMS AND CONDITIONS

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossary

Signalling Data Link Service Data Unit
: A grouping of SDL user data whose boundaries are preserved from one end of the signalling data link connection to the other.

Data transfer
: The phase in connection and connectionless modes that supports the transfer of data between to signalling data link users.

SDL provider
: The signalling data link layer protocol that provides the services of the signalling data link interface.

SDL user
: The user-level application or user-level or kernel-level protocol that accesses the services of the signalling data link layer.

Local management
: The phase in connection and connectionless modes in which a SDL user initializes a stream and attaches a PPA address to the stream. Primitives in this phase generate local operations only.

PPA
: The point at which a system attaches itself to a physical communications medium.

PPA identifier
: An identifier of a particular physical medium over which communication transpires.

Acronyms

ITU-T	International Telecommunications Union - Telecom Sector
LM	Local Management
LMS	Local Management Service
LMS Provider	A provider of Local Management Services
LMS User	A user of Local Management Services
PPA	Physical Point of Attachment
SDL	Signalling Data Link
SDL SDU	Signalling Data Link Service Data Unit
SDLI	Signalling Data Link Interface
SDLS	Signalling Data Link Service
SDT	Signalling Data Terminal
SDTI	Signalling Data Terminal Interface
SDTS	Signalling Data Terminal Service
SL	Signalling Link
SLI	Signalling Link Interface
SLS	Signalling Link Service
SS7	Signalling System No. 7

References

[1] ITU-T Recommendation Q.700
[2] ITU-T Recommendation Q.701
[3] ITU-T Recommendation Q.702
[4] ITU-T Recommendation Q.703
[5] ITU-T Recommendation Q.704
[6] Geoffrey Gerrien, “CDI - Application Program Interface Guide,” Gcom, Inc., March 1999.
[7] ITU-T Recommendation Q.771

Indices

Concept Index

L

license, FDL 149
license, GNU Free Documentation License 149

S

STREAMS 1, 3, 4, 5

Type Index

L

lmi_attach_req_t 39
lmi_detach_req_t 42
lmi_disable_con_t 53
lmi_disable_req_t 50
lmi_enable_con_t 49
lmi_enable_req_t 45
lmi_error_ack_t 29
lmi_error_ind_t 60
lmi_event_ind_t 65
lmi_info_ack_t 37
lmi_info_req_t 34
lmi_ok_ack_t 27
lmi_optmgmt_ack_t 58, 124
lmi_optmgmt_req_t 54, 120
lmi_stats_ind_t 64

S

sl_bsnt_ind_t 95
sl_bsnt_not_retr_ind_t 96
sl_clear_buffers_req_t 104
sl_clear_rtb_req_t 106
sl_cong_accept_req_t 89
sl_cong_discard_req_t 87
sl_continue_req_t 118
sl_emergency_ceases_req_t 70
sl_emergency_req_t; 68
sl_in_service_ind_t 75
sl_link_cong_ceased_ind_t 85
sl_link_cong_ind_t 83
sl_loc_proc_out_ind_t 112
sl_local_proc_outage_req_t 110
sl_no_cong_req_t 91
sl_out_of_service_ind_t 76
sl_pdu_ind_t 82
sl_pdu_req_t 80
sl_power_on_req_t 66
sl_rb_cleared_ind_t 108
sl_rem_proc_out_ind_t 116
sl_rem_proc_recovered_ind_t 117
sl_resume_req_t 113
sl_retrieval_comp_ind_t 101
sl_retrieval_not_poss_ind_t 103
sl_retrieval_req_and_fsnc_t 97
sl_retrieve_bsnt_req_t 93
sl_retrieved_msg_ind_t 99
sl_rtb_cleared_ind_t 109
sl_start_req_t 72
sl_stop_req_t 78

Variable Index

L

lmi_correct_primitive 27
lmi_errno 29, 31, 60, 62
lmi_error_primitive 31
lmi_header_len 38
lmi_interval 64
lmi_max_sdu 38
lmi_mgmt_flags 54, 58, 59, 120, 124, 125
lmi_min_sdu 38
lmi_objectid 65
lmi_opt_length 54, 58, 120, 124
lmi_opt_offset 54, 58, 120, 124
lmi_ppa 39
lmi_ppa_addr 38
lmi_ppa_style 38, 39, 42
lmi_primitive .. 27, 29, 34, 37, 39, 42, 45, 49, 50, 53, 54, 58, 60, 64, 65, 120, 124
lmi_reason 31, 62
lmi_rem 45
lmi_severity 65
lmi_state 27, 32, 37, 49, 53, 62
lmi_timestamp 64, 65
lmi_version 37

S

sl_bsnt 95, 96
sl_cong_status 83, 85
sl_disc_status 83, 85
sl_fsnc 97
sl_mp 80, 82, 99, 101
sl_primitive 66, 68, 70, 72, 75, 76, 78, 80, 82, 83, 85, 87, 89, 91, 93, 95, 96, 97, 99, 101, 103, 104, 106, 108, 109, 110, 112, 113, 115, 116, 117, 118, 126, 128
sl_reason 76
sl_timestamp 76, 83, 85, 112, 115, 116, 117

(Index is nonexistent)

Primitive Index

L

LMI_ATTACH_REQ .. 9, 10, 11, 27, 31, 32, 37, 38, 39, 62
LMI_DETACH_REQ 9, 10, 11, 27, 31, 42
LMI_DISABLE_CON 13, 32, 33, 38, 50, 53, 63
LMI_DISABLE_REQ 9, 13, 32, 50
LMI_ENABLE_CON 12, 32, 37, 46, 49, 63
LMI_ENABLE_REQ ... 9, 12, 28, 31, 32, 37, 45, 53, 63
LMI_ERROR_ACK 9, 11, 12, 13, 29, 32, 34, 39, 42, 46, 50, 55, 67, 68, 70, 72, 78, 88, 89, 91, 94, 98, 104, 106, 110, 113, 118, 121, 126, 129
LMI_ERROR_IND 14, 32, 60
LMI_ERRORK_ACK 12, 13
LMI_EVENT_IND 15, 32, 65, 126
LMI_INFO_ACK 10, 32, 34, 37, 39, 42
LMI_INFO_REQ 9, 10, 31, 34, 37
LMI_OK_ACK 9, 11, 27, 32, 39, 42
LMI_OPTMGMT_ACK 13, 32, 55, 58, 120
LMI_OPTMGMT_REQ ... 9, 13, 32, 54, 58, 59, 120, 124
LMI_STATS_IND 14, 32, 64

M

M_DATA 80, 82, 99, 101, 102
M_PCPROTO ... 27, 29, 34, 37, 38, 54, 58, 68, 70, 72, 76, 78, 83, 85, 87, 89, 91, 93, 95, 96, 97, 104, 106, 108, 110, 113, 118, 120, 124
M_PROTO .. 30, 34, 35, 37, 38, 39, 40, 42, 43, 45, 47, 49, 50, 51, 53, 54, 56, 60, 61, 64, 65, 66, 67, 68, 69, 70, 71, 72, 74, 75, 76, 78, 79, 80, 82, 83, 85, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 101, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 122, 127

S

SL_BSNT_IND 20, 94, 95
SL_BSNT_NOT_RETRIEVABLE_IND 20, 94, 96
SL_CLEAR_BUFFERS_REQ 23, 104, 108, 109
SL_CLEAR_RTB_REQ 23, 106, 109
SL_CONGESTION_ACCEPT_REQ 19, 87, 89
SL_CONGESTION_CEASED_IND 85
SL_CONGESTION_DISCARD_REQ 19, 87
SL_CONTINUE_REQ 25, 118
SL_EMERGENCY_CEASES_REQ 16, 70
SL_EMERGENCY_REQ 16, 68
SL_IN_SERVICE_IND 16, 72, 75
SL_LINK_CONGESTED_IND 19, 83
SL_LINK_CONGESTION_CEASED_IND 19, 84, 86
SL_LINK_CONGESTION_IND 84, 86
SL_LOCAL_PROCESSOR_OUTAGE_IND 24, 112, 115
SL_LOCAL_PROCESSOR_OUTAGE_REQ 24, 110, 113
SL_LOCAL_PROCESSOR_RECOVERED_IND 24, 112, 115
SL_NO_CONGESTION_REQ 19, 91
SL_NOTIFY_IND 26, 128
SL_NOTIFY_REQ 26, 126, 128
SL_OPTMGMT_ACK 25, 121, 124
SL_OPTMGMT_REQ 25, 120, 124, 125
SL_OUT_OF_SERVICE_IND 17, 72, 76
SL_PDU_IND 18, 82
SL_PDU_REQ 18, 80, 99, 101
SL_POWER_ON_REQ 15, 66
SL_RB_CLEARED_IND 23, 104, 108
SL_REMOTE_PROCESSOR_OUTAGE_IND ... 25, 116, 117
SL_REMOTE_PROCESSOR_RECOVERED_IND 25, 116, 117
SL_RESUME_REQ 24, 113
SL_RETREIVAL_REQUEST_AND_FSNC_REQ 103
SL_RETRIEVAL_COMPLETE_IND 21, 97, 101
SL_RETRIEVAL_NOT_POSSIBLE_IND 22, 98, 103
SL_RETRIEVAL_REQ_AND_FSNC_REQ 97
SL_RETRIEVAL_REQUEST_AND_FSNC_REQ 21, 99, 101, 102, 103
SL_RETRIEVE_BSNT_REQ 20, 93, 95, 96
SL_RETRIEVED_MESSAGE_IND 21, 97, 99, 102
SL_RTB_CLEARED_IND 23, 104, 106, 109
SL_START_REQ 16, 72, 75, 77
SL_STOP_REQ 17, 78, 93

Primitive Value Index

L

LMI_CHECK 54, 59, 121, 125
LMI_CURRENT 55, 59, 121, 125
LMI_DEFAULT 55, 59, 121, 125
LMI_FAILURE 58, 124
LMI_NEGOTIATE 54, 59, 120, 125
LMI_NOTSUPPORT 59, 125
LMI_PARTSUCCESS 58, 124
LMI_READONLY 59, 125
LMI_STYLE1 38
LMI_STYLE2 ..... 38, 39, 42
LMI_SUCCESS 58, 124

S

SL_FAIL_ABNORMAL_BSNR 76
SL_FAIL_ABNORMAL_FIBR 76
SL_FAIL_ACK_TIMEOUT 76
SL_FAIL_ALIGNMENT_NOT_POSSIBLE 77
SL_FAIL_CONG_TIMEOUT 76
SL_FAIL_RECEIVED_SIE 77
SL_FAIL_RECEIVED_SIN 77
SL_FAIL_RECEIVED_SIO 77
SL_FAIL_RECEIVED_SIOS 77
SL_FAIL_SUERM_EIM 76
SL_FAIL_T1_TIMEOUT 77
SL_FAIL_UNSPECIFIED 76

Protocol State Index

L

LMI_ATTACH_PENDING 28, 32, 37, 39, 62
LMI_DETACH_PENDING 28, 33, 38, 42, 63
LMI_DISABLE_PENDING 33, 38, 50, 53, 63
LMI_DISABLED 10, 28, 32, 37, 39, 42, 45, 50, 53, 63
LMI_ENABLE_PENDING 32, 37, 45, 49, 63
LMI_ENABLED 28, 32, 38, 46, 49, 50, 63, 65, 66, 67, 68, 70, 72, 75, 77, 78, 80, 82, 83, 84, 85, 86, 87, 88, 89, 91, 93, 94, 95, 96, 97, 98, 99, 101, 103, 104, 106, 108, 109, 110, 112, 113, 115, 116, 117, 118
LMI_UNATTACHED 10, 27, 28, 32, 37, 39, 42, 62
LMI_UNUSABLE 28, 32, 37, 63

S

SL_STATEPOWER_OFF 72
SL_STATE_ALIGNED_NOT_READY 78
SL_STATE_ALIGNED_READY 75, 78
SL_STATE_IN_SERVICE ... 72, 75, 77, 78, 80, 82, 83, 84, 85, 86, 87, 88, 89, 91, 113, 115, 118
SL_STATE_INITIAL_ALIGNMENT 72, 78
SL_STATE_OUT_OF_SERVICE .. 66, 72, 75, 77, 78, 93, 94, 95, 97, 98, 99, 101, 104, 106, 108, 109
SL_STATE_POWER_OFF 66, 67, 77, 78
SL_STATE_PROCESSOR_OUTAGE .. 110, 112, 113, 115, 116, 117, 118

Protocol Error Index

L

LMI_BADADDRESS 29, 34, 40, 42, 46, 50, 55, 60, 121
LMI_BADADDRTYPE 29, 35, 40, 43, 46, 51, 55, 60, 121
LMI_BADDIAL 29, 35, 40, 43, 46, 51, 55, 60, 121
LMI_BADDIALTYPE 29, 35, 40, 43, 46, 51, 55, 60, 121
LMI_BADDISPOSAL 29, 35, 40, 43, 46, 51, 55, 61, 122
LMI_BADFRAME ... 29, 35, 40, 43, 46, 51, 55, 61, 122
LMI_BADPPA 29, 35, 40, 43, 46, 51, 56, 61, 122
LMI_BADPRIM 30, 35, 40, 43, 46, 51, 56, 61, 122
LMI_BUSY 30, 36, 41, 44, 47, 52, 56, 62, 123
LMI_CALLREJECT 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_CRCERR 30, 35, 41, 43, 47, 51, 56, 61, 122
LMI_DEVERR .. 31, 36, 41, 44, 48, 52, 57, 62, 67, 69, 71, 74, 79, 88, 90, 92, 94, 98, 105, 107, 111, 114, 119, 123, 127
LMI_DISC 30, 35, 40, 43, 46, 51, 56, 61, 67, 73, 94, 98, 105, 106, 111, 113, 119, 122, 126
LMI_DLE_EOT 30, 35, 41, 43, 47, 51, 56, 61, 122
LMI_DSRTIMEOUT 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_EVENT ... 30, 35, 40, 43, 46, 51, 56, 61, 67, 68, 70, 73, 79, 88, 89, 91, 94, 98, 105, 107, 111, 113, 119, 122, 126
LMI_FATALERR 30, 35, 40, 43, 46, 51, 56, 61, 67, 68, 70, 73, 79, 88, 90, 92, 94, 98, 105, 107, 111, 114, 119, 122, 127
LMI_FORMAT 30, 35, 41, 43, 47, 51, 56, 61, 122
LMI_HDLC_ABORT 30, 36, 41, 44, 47, 52, 56, 61, 122
LMI_HDLC_IDLE .. 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_HDLC_NOTIDLE ... 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_INCOMPLETE 30, 36, 41, 44, 47, 52, 56, 62, 123
LMI_INITFAILED 30, 35, 40, 43, 46, 51, 56, 61, 67, 122
LMI_LAN_COLLISIONS 31, 36, 41, 44, 48, 52, 57, 62, 123
LMI_LAN_NOSTATION .. 31, 36, 41, 44, 48, 52, 57, 62, 123
LMI_LAN_REFUSED 31, 36, 41, 44, 48, 52, 57, 62, 123
LMI_LOSTCTS 31, 36, 41, 44, 48, 52, 57, 62, 123
LMI_NOANSWER ... 30, 36, 41, 44, 47, 52, 57, 62, 123
LMI_NOTSUPP 30, 35, 40, 43, 46, 51, 56, 61, 122
LMI_OUTSTATE 30, 35, 40, 43, 47, 51, 56, 61, 67, 69, 71, 74, 79, 88, 90, 92, 94, 98, 105, 107, 111, 114, 119, 122, 127
LMI_OVERRUN 30, 36, 41, 44, 47, 52, 56, 61, 122
LMI_PROTOSHORT 30, 35, 40, 43, 47, 51, 56, 61, 67, 69, 71, 74, 79, 88, 90, 92, 94, 98, 105, 107, 111, 114, 119, 122, 127
LMI_QUIESCENT .. 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_RESUMED 31, 36, 41, 44, 47, 52, 57, 62, 123
LMI_SYSERR .. 30, 31, 35, 40, 43, 47, 51, 56, 61, 62, 67, 69, 71, 74, 79, 88, 90, 92, 94, 98, 105, 107, 111, 114, 119, 122, 127
LMI_TOOSHORT ... 30, 36, 41, 44, 47, 52, 56, 61, 122
LMI_UNSPEC .. 29, 34, 40, 42, 46, 50, 55, 60, 67, 68, 70, 73, 78, 88, 89, 91, 94, 98, 104, 106, 111, 113, 118, 121, 126
LMI_WRITEFAIL .. 30, 35, 40, 43, 47, 51, 56, 61, 122

S

SL_FAIL_ABNORMAL_BSNR 73
SL_FAIL_ABNORMAL_FIBR 73
SL_FAIL_ACK_TIMEOUT 73
SL_FAIL_ALIGNMENT_NOT_POSSIBLE 73
SL_FAIL_CONG_TIMEOUT 73
SL_FAIL_RECEIVED_SIE 73
SL_FAIL_RECEIVED_SIN 73
SL_FAIL_RECEIVED_SIO 73
SL_FAIL_RECEIVED_SIOS 73
SL_FAIL_SUERM_EIM 73
SL_FAIL_T1_TIMEOUT 73
SL_FAIL_UNSPECIFIED 73

Manual Page Index

C

close(2) 10

E

errno(3) 31, 62

O

open(2) 10, 38